User Reference Manual

C3Dserver

By Motion Lab Systems, Inc.

This manual was written by Motion Lab Systems using *ComponentOne Doc-To-Help*.™

Updated Monday, July 19, 2010

Intended Audience

This manual is written to provide a general description and usage guidelines for anyone using this application – it does not provide any clinical interpretation of the data that you may collect and analyze and no clinical interpretations should be assumed.

Operating Environment

This program runs on any 32-bit Microsoft operating system using an Intel compatible processor. This application should also run under current versions of WINE, an Open Source implementation of the Windows API on top of X and Unix that runs on Linux and FreeBSD operating systems, although full operation is not guaranteed under WINE.

It is assumed that the end-user is familiar with the operating system environment that they are using and no special reference is made to any specific operating system within this manual. Manuals for these operating systems are available from the appropriate sources – contact your supplier or system administrator if you need additional support for your operating system.

All Motion Lab Systems applications fully support the C3D file format. Detailed information on the C3D file format is available on the Internet at http://www.c3d.org - additional information on manufacturer specific C3D implementations may be obtained from your C3D application developer.

Year 2000 compliance

Motion Lab Systems, Inc. has reviewed and tested this application for Year 2000 (Y2K) compliance. The program will continue to function correctly on and beyond the year 2000.

Trademarks

All trademarks and registered trademarks are the property of their respective owners.

Motion Lab Systems, Inc.
15045 Old Hammond Highway • Baton Rouge, LA 70816-1244
Phone (225) 272-7364 • Fax (225) 272-7336
Email: support@motion-labs.com
http://www.motion-labs.com

Printed in the United States of America

Contents

The C3Dserver

The C3Dserver is a Dynamic Link Library, for Microsoft Windows based operating systems. The C3Dserver is written and maintained by Motion Lab Systems to provide programming easy access to the C3D file format. Macros and applications written using the C3Dserver can open and close, read and write, and modify any version of the C3D file format for all files that conform to the C3D file specification described in the C3D User Guide.

The C3D file format is a public domain format documented at www.c3d.org where user manuals, format descriptions, applications and sample data files are available free of charge. We strongly recommend that anyone using, or programming, with the C3Dserver read the C3D file documentation available from the C3D web site.

C3D files are used throughout the biomechanics and motion capture markets to record 3D and Analog data, together with necessary information about the data to accurately interpret the data. C3D files can be easily exchanged between both clinical laboratories and researchers with the stability and extendibility of file format ensuring that it is as easy to read biomechanics data recorded today, as data recorded twenty years ago.

Overview

The C3D (**C**oordinate **3D,** pronounced *see-three-dee*) data file format was developed at the National Institutes of Health in 1987 and conforms to a publicly available C3D file specification. The format description of the C3D file structure, and a detailed programmer's manual, is available at no charge at http://www.c3d.org and should be downloaded and read by anyone using the C3Dserver.

The design of the C3D file format was originally driven by the need for a convenient and efficient format to store and interpret biomechanics data. The C3D format stores 3D coordinate and numeric data for any measurement trial, with all the various parameters that describe the data, in a single file. This largely eliminates the need for the biomechanics trial data (recording the 3D motion, forces, electromyography etc.) to always travel around with additional notes and subject information to describe the data. With most conventional and manufacturer created file formats these additional descriptive information usually gets separated from the data at some point in its travels – this can not happen with C3D as a single file can contain all the information necessary to describe the data within the C3D file.

The ability to store a multitude of information ***about*** the data is the feature that sets the C3D format apart from every other biomechanics format. Early in the design of the C3D format it was realized that it was unlikely that one, ironclad, specification

would fit every biomechanics need. As a result, the C3D file usually stores a small number of common parameters that describe the 3D data and then allows the users to define, generate, and store within the file any number of user or lab defined data items so that anyone opening the C3D file can access them.

C3Dserver SDK

A comprehensive manual that documents the C3D file format and structure is available for download from the c3d web site. The C3D file format is in the public domain.

The Motion Lab Systems C3Dserver is supplied with a Software Development Kit (SDK) that provides an easy way for anyone to access and manipulate any data stored in the C3D file structure. The C3Dserver allows complete access to edit and change the data or parameters in any C3D file. The C3Dserver is designed to work within the 32-bit Microsoft windows environment using COM via a simple, well-defined, interface with well documented functions and includes fully functional example source code for Visual Basic and C++ using Microsoft Foundation Classes. The C3Dserver eliminates the need for any programmer to write code to support C3D files at a binary level.

The C3Dserver is supplied with complete source code to several applications written in Visual Basic and C++ that demonstrate the features and ease of use of the C3Dserver – these sample applications provide an easy way for students and users to explore the C3Dserver and C3D data access.

In addition to providing the basic functionality of C3D file access, the C3Dserver SDK includes a sample Excel spreadsheet written using Visual Basic macros. Using these examples, most users can write programs to perform C3D data access within three to four hours of installing the C3Dserver.

Finally, the C3Dserver is available as a fully functional evaluation version that does not expire and may be used non-commercially at non-charge. The only difference between the evaluation version and the commercial version is the application execution speed. Applications written with the C3Dserver will run in any 32-bit Microsoft Windows environment.

The C3Dserver is available free of charge for all users – both commercial and non-commercial. It is supplied as a free version and a faster licensed version – the free version may be used to develop commercial applications allowing the end-user to purchase a C3Dserver license directly for Motion Lab Systems if they need higher performance or simply wish to contribute to further development of the C3Dserver.

Features

The C3Dserver enables anyone to jump-start the process of accessing data in the C3D file structure since it is now accessible via Visual Basic, C++, Java or any other COM supported language. All C3D access functions are now in the server so the user can concentrate of reading parameters and data without worrying if the information is floating-point or integer storage or INTEL (MS-DOS), SGI or DEC format. This means that anyone using the C3Dserver can simply open any C3D file and read the information without worrying about the format or storage methods. The C3Dserver accesses C3D files produced by any manufacturer at a binary level with complete transparency to the programmer.

In addition, other novel applications can be quickly implemented using the C3Dserver. For instance, it is quite simple to write an application in Java to allow a web site to read a C3D file and provide remote analysis or diagnostic information. Unlike conventional program libraries, the C3Dserver allows the application

specialist to concentrate on providing functionality to the user instead of worrying about the details of reading the file structure and data format.

Licensing

The C3Dserver DLL is available as a free evaluation version that contains all the functionality of the licensed version and runs at an adequate speed for general student and many commercial programming applications. The evaluation version does not expire and is available at no charge but is technically unsupported – this means that Motion Lab Systems makes no commitment to respond to problems or issues surrounding the use of the C3Dserver in this environment.

In addition to the free evaluation version, a high speed, fully optimized version of the C3Dserver is available commercially – applications that may take five or ten seconds to execute with the evaluation version will appear to complete almost instantly with the licensed version of the C3Dserver:

- Evaluation version – The unlicensed version does not run at full speed but may be used to develop any number of applications for both commercial and non-commercial use. This version is unsupported.
- Licensed version - a fully optimized version of the C3Dserver licensed for use at a single location. This version includes full product support to the licensed location via phone and email for one year from the date of purchase.

Other than the speed of operation, there is no other difference between the two versions of the C3Dserver. Applications written for one version will function in exactly that same way on the evaluation version as the licensed version except for the speed of operation.

Commercial applications developed with either version of the C3Dserver may be distributed with the free evaluation version of the C3Dserver provided that the use of the C3Dserver is documented and credited to Motion Lab Systems, Inc. End-users of applications developed with the C3Dserver may license a copy of the C3Dserver from Motion Lab Systems to enable the C3Dserver to run at full speed at their location.

Licensed versions of the C3Dserver are registered to specific locations and will have a unique User Name and Organization associated with each licensed copy – this restricts the physical location of the licensed C3Dserver but does not limit the number of copies of the C3Dserver that may be in use at the licensed locations.

Performance

All applications written using the C3Dserver will run faster when a license is purchased.

The only difference in performance between the free evaluation version and a fully licensed version of the C3Dserver is that each function call by the evaluation version incurs short delay that is removed when the software is purchased and the license is installed. All C3Dserver function calls perform identically except for the delay in the free evaluation version.

Motion Lab Systems, Inc., plans to keep the C3Dserver up to date with the emerging C3D standard - thus any application developed using the C3Dserver would have access to any format changes and/or enhancements to the C3D format without requiring a complete rewrite of the applications basic file access functions. This offers a huge advantage to application programmers who will generally not have to rewrite software applications to accommodate any changes that may occur in the C3D format as it evolves and additional enhancement are added in the future.

Using the C3Dserver

The C3Dserver is a Microsoft COM component that allows you to write programs, which will give you access to all the data in a C3D file. Microsoft COM (Component Object Model) technology in the Microsoft windows Operating Systems enables software components to communicate. COM allows you to create re-usable software components, link components together to build applications, and take advantage of Windows services. The family of COM technologies includes COM+, Distributed COM (DCOM) and ActiveX® Controls.

A COM server consists of a collection of interfaces – in this case, there is only one interface at this time. An interface is a way for the object to expose its functionality to other programs that use it. In COM, the interface is really a table of pointers to the available functions of the object.

Note that much of the information provided about specific calls to the C3Dserver assumes that the programmer understands the C3D file format. A full description of the C3D file format is available from the C3D web site at http://www.c3d.org and in the companion documentation "The C3D Format – A Technical User Manual" available from Motion Lab Systems and from the C3D web site at no charge. It is assumed that anyone using the C3Dserver has access to this documentation resource.

C3DServer Installation

The C3Dserver is implemented in C++ as a 32-bit Dynamic Link Library or DLL and is automatically installed in your system directory. The DLL is automatically registered with the operating system when it is installed and can be used immediately without requiring a reboot.

All sample code supplied with the C3Dserver is installed in subdirectories of the **C:\Program Files\Motion Lab Systems\C3Dserver** root by default. Additional MFC dynamic link libraries required by the sample applications will be installed if you select the example applications.

The typical C3Dserver installation will create a program group called “Motion Lab Systems” with shortcuts to the C3Dserver documentation as well as the sample applications supplied with the C3Dserver. Additional sample C3D files can be downloaded by anonymous ftp from the C3D ftp site at ftp.c3d.org or directly from the Motion Lab Systems web site at http://www.motion-labs.com.

The “Motion Lab Systems” program group contains an additional folder called “Utilities” that contains links to the Motion Lab Systems web pages and technical

support pages, as well as a “Check for updates” link that can use the Internet to automatically check for program updates, then download and install them.

The full installation includes a software test harness (written in Visual Basic, source code included), along with two sample C3D data files (additional sample C3D files are available from the C3D ftp site) and full source code and example data access programs using Visual Basic, C++ and Excel. A simple C3D file editor/viewer has been written using the C3Dserver interface - this is included together with complete Visual Basic source code listings.

In addition, a sample C++ application (complete with full source code) is provided to illustrate the creation of C3D files in any of the supported formats.

The C3Dserver can be completely removed from your system at any time by selecting "Add/Remove" from the Control Panel.

Standalone Installation

If you want to install the C3Dserver simply as a DLL so that it can be accessed as a system resource by other programs without installing the associated demo files that are supplied with the C3Dserver then all you need to do is copy the C3Dserver.dll to an appropriate directory and register it with the operating system. The standard C3Dserver installation copies the DLL to the operating system dynamic link library directory and we recommend that you do the same. The C3Dserver can be registered by issuing the following command at the operating system prompt:

```
regsrv32 c:\windows\system32\c3dserver.dll
```

Verifying the C3Dserver functions

You can verify the correct C3Dserver installation at time by running the VB Function Test at any time.

The full C3Dserver installation includes a software test harness application called *vbservertest.exe* that exercises all of the C3Dserver functions, reporting back on the results of the tests. The test harness application is written in Visual Basic and includes the full VB source code so that anyone can see exactly how each of the C3Dserver functions is used and tested. It is installed when the C3Dserver is installed and can be run as the *VB Function Test* application from the C3Dserver menu.

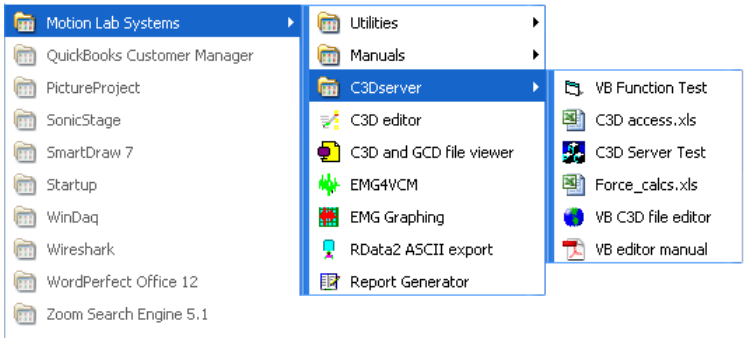

Figure 1 - The C3Dserver is installed with several useful tools

In addition to the *VB Function Test*, the C3Dserver is installed with a number of useful applications and utilities. The ability to use the C3Dserver with common Microsoft applications is demonstrated by the *C3D access.xls* spreadsheet which accesses the C3Dserver via Visual Basic macros in the spreadsheet. The *C3D Server Test* application is an application written in C++ that uses the C3Dserver to create a simple C3D file filled with example data – full C++ source code is included for this

application to demonstrate how the C3Dserver is called and functions with the C++ compiler.

The VB C3D file editor is a very useful C3D utility written in Visual Basic and includes full source code for the project.

In addition, a simple C3D file editor is installed as *VB C3D file editor*. This is a simple graphical editor for C3D files written in Visual Basic to demonstrate the functions provided by the C3Dserver. It is supplied free of charge and includes the full source code, written in Visual Basic to allow it to be modified and recompiled if desired. The *VB C3D file editor* can be used to open and view c3D files and is a useful test if you have problems opening C3D files with the C3Dserver as the *VB C3D file editor* uses the C3Dserver functions to open and edit C3D files in the same way that other applications use the C3Dserver.

Note that the VB C3D file editor makes extensive use of the C3Dserver functions so if you are using the free standard evaluation version of the C3Dserver it may be very slow when performing some functions. However, since it only uses C3Dserver functions to access the files that it opens it is an excellent test if you experience problems processing C3D files with your own applications.

The *VB C3D file editor* includes a brief manual that describes the editor functions. The C3Dserver Manual and C3D User Guide are installed separately in the Manuals menu together with the manuals and documentation for other Motion Lab Systems products.

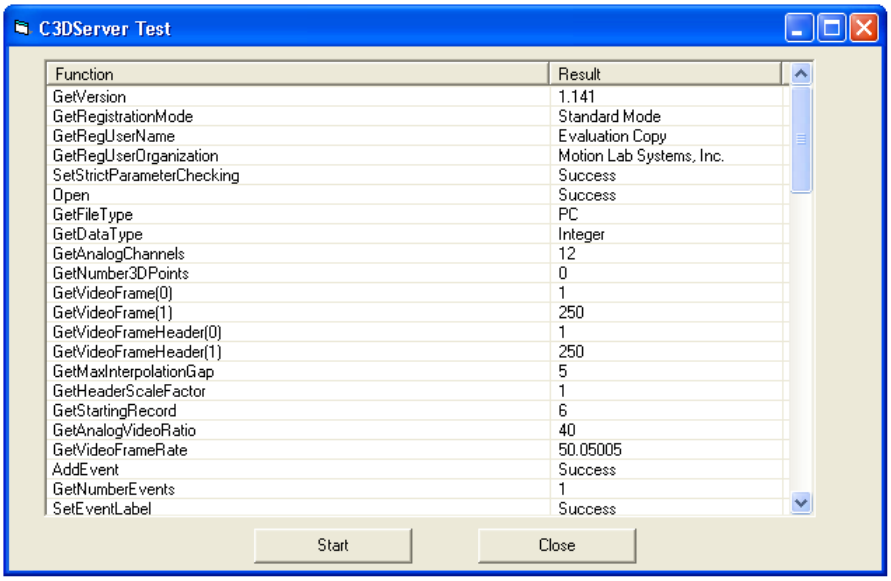

Figure 2 - The VB Function Test has a user friendly graphical interface.

The *VB Function Test* has a simple GUI interface that displays the names of each of the function calls as it is performed and the results of the tests. This makes the *VB Function Test* an excellent debugging tool in case you run into problems with either the C3Dserver or any of the C3D files that you are using.

Note the some of the first functions display the version number of the installed copy of the C3Dserver and the Registration mode … either the *Standard Version* (free for evaluation but with a small delay each time a function is called), or the *Professional Version* which runs at full speed. The *VB Function Test* also displays the registered user name and user organization so that a valid registration can be confirmed.

Programming using Visual C++

The following steps need to be taken to write a program using Visual C++ that access C3D files via the C3Dserver.

- Start a new project that you want to use.
- Make sure that the COM Library is initialized using CoInitialize or CoInitializeEx. The COM library should be closed using CoUnitialize or CoUninitializeEx.
- Import the c3dserver.dll file into the source file where it is going to reference the server using the #import "c3dserver.dll" directive. This directive creates wrapper classes for the type library. When you compile the project, you will find that there is a c3dserver.tlh (the header for the wrapper classes) and c3dserver.tli (the implementation for the wrapper classes) file in the target directory.

You can then get a smart pointer to the interface by a call shown below:

```
C3DSERVERLib::IC3DPtr p(_uuidof(C3DSERVERLib::C3D));
```

This pointer **p** can then be used to access all the functions exposed by the server. Note that if you are using smart pointers, you do not need to call Release because the pointer will handle that.

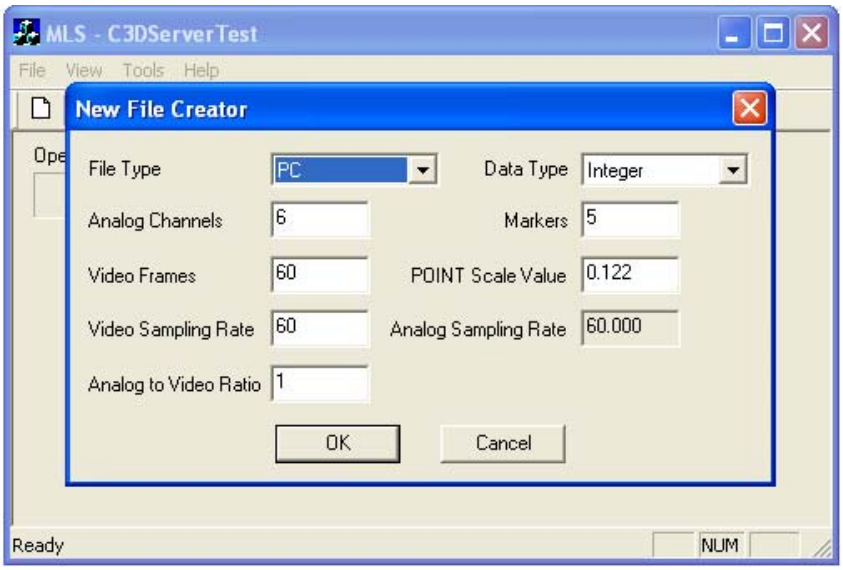

Figure 3 - A sample C++ application creates C3D files containing random data.

Full source code for an example Visual C++ application is supplied with the C3Dserver. The sample application creates a C3D file containing random 3D and analog data within ranges that you specify and in any of the supported C3D formats. This can serve as a convenient test-bed to create C3D files and any of the supported file formats to test other applications.

In addition, the supplied sample code can be used as a framework that can be extended to create individual applications. Applications written using the C3Dserver are capable of running under any 32-bit Microsoft operating system – the supplied example code has been tested under Windows 98, Windows NT, Windows 2000, and Windows XP-Professional

Programming using Visual Basic

The following steps need to be taken to write a program using Visual Basic that accesses C3D files via the C3Dserver.

- Start a new project, either a complex one or a single form project.
- Go to the Project – References menu item. This will open a dialog box that will show all the available references for the project. Select the entry labeled as **C3DServer 1.0 Type Library**. This entry should already be present in the list (but unchecked), if you have registered the server. If it is not present in the list, you can click on the Browse button, look for the C3DServer.dll file, and select it.
- See the section C3DServer Module for details on how to get the interface needed and access the functions.

Writing a macro using Excel is very similar to the procedure described above. When you are in the macro editor, you need to select the **Tools – References** menu item and you can see the list of references. Once that is done the other things are just like in Visual Basic.

Complete source code is supplied for a sample Visual Basic application that displays and edits the contents of any C3D file. This is a both useful starting point for novice programmers and also an extremely useful tool that allows anyone to manually edit a C3D that they may have just created to correct small errors and test it without the time and effort of changing their program and recompiling just to play "what if?"

You can use the Visual Basic C3D file editor that is included with the C3Dserver to open any C3D file and view the contents of the file. The editor will allow you to examine and change header event information, 3D data values, analog data values, and the contents of any of the parameters stored in the C3D file. Parameter data is displayed in a common tree format that allows the user to easily determine the group and parameter relationships within the C3D file. Naturally, the results of any edits within the C3D file can be saved to disk, allowing the programmer to quickly resolve formatting and data issues that may need attention when developing specific C3D applications.

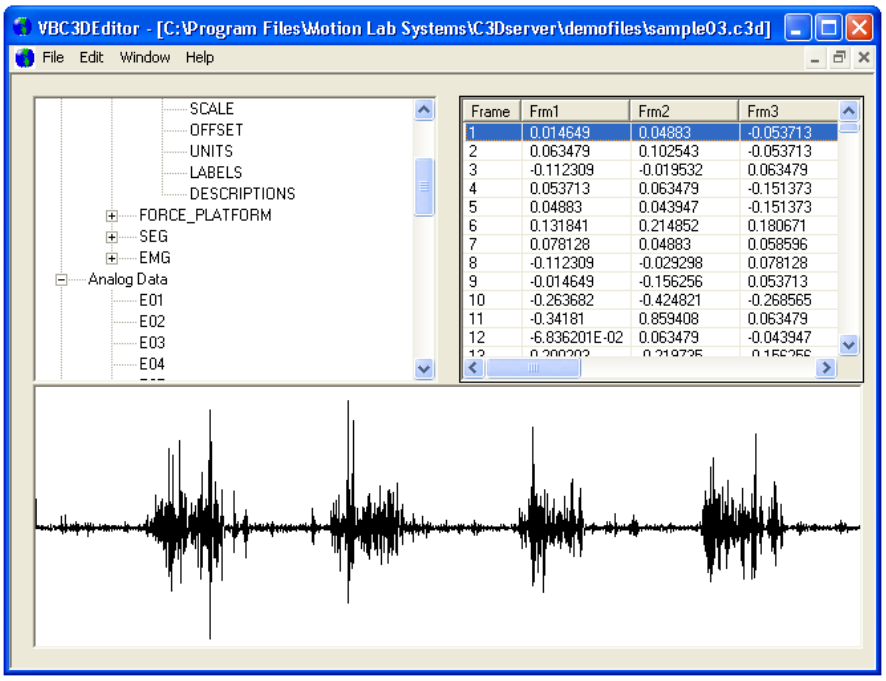

Figure 4 - A sample Visual Basic C3D file editor is supplied with the C3Dserver.

In addition, since the full Visual Basic source code for this C3D editor is available for study, it is quite easy for a programmer to extract code fragments from the C3D file editor to incorporate into other applications. There are no licensing or copyright issues to worry about when reusing this code as Motion Lab Systems encourages programmers to use the supplied examples within any other C3Dserver application.

In addition, careful study of the source code supplied with both the Visual Basic C3D editor and the C++ C3D file generator helps new programmers to understand how the C3Dserver is used. Both of the sample applications use large numbers of the functions supported by the C3Dserver.

The Visual Basic C3D file editor, supplied with the C3Dserver, is a simplified version of the C3Deditor, a commercial application written by Motion Lab Systems that provides many more features including filtering, batch data processing and C3D file validity checking. A demonstration version of the commercial version of the C3Deditor is available for evaluation and purchase from the Motion Lab Systems web site at http://www.motion-labs.com. Many serious C3D applications programmers eventually purchase the full featured C3Deditor which can be used to repair and diagnose almost any C3D formatting issue.

C3Dserver Module

The C3Dserver module allows you to write programs and retrieve data from any C3D file. The C3Dserver provides all of the functions needed to access or modify the data in the C3D file.

You must create an instance of the server first and then you can open the file using this interface.

You need to declare the interface reference using the command below. The variable `itf` is declared as being of `type As C3DSERVERLib.C3D`. You can do this

outside of all the functions and procedures so that this variable is available in all of them.

```
Dim itf As C3DSERVERLib.C3D
```

You then need to create an instance of the object. This is done using the command shown below. Here a C3DSERVER object is created and you are given a reference to the interface C3D, which is the one you will use to access the functions.

```
Set itf = New C3DSERVERLib.C3D
```

This interface can now be used to open and close files. You must first open a file successfully before you can call any of the other functions.

As example, this is how you can open a file.

```
On Error GoTo err_Generic
 Dim nReturn As Integer
 nReturn = itf.Open(sFilename,3)
 Exit Function
 err_Generic:
 Dim str1 As String
 str1 = CStr(itf.GetLastError) + ":" + Err.Description
 MsgBox str1
```

The code snippet above shows how to open a file. In all functions that you call you must try to catch any errors that might be reported. Once you catch the error, you can get the description using `Err.Description`. Alternatively, you can call the `GetLastError` function. This will return an integer and you can provide your own custom description based on this value. The return error values are listed at the end of this manual.

When you are done using the interface after closing the file and you have no use for it, you should use the command,

```
Set itf = Nothing
```

This will free up the resources associated with the object.

Testing that the C3Dserver is installed

A typical function to check whether the server exists is shown below written using VB.NET code. The idea is that you just instantiate the C3DServer and if that fails, you can assume that the server is not present or has not been registered correctly.

This function should be the first thing that is called when the program starts. If this function call fails, then your application can close gracefully and the user can install the C3Dserver.

```
Private Function DoesServerExist() As Boolean
    Dim c3d As C3DSERVERLib.C3D
    Try
        c3d = New C3DSERVERLib.C3D
    Catch ex As Exception
        MessageBox.Show("C3Dserver is not installed")
```

```
            Return False
        End Try
        c3d = Nothing
        Return True
    End Function
```

C3Dserver Functions

The C3Dserver includes several functions that provide information about the C3Dserver – allowing the user to determine the current version number, the status (license type) and licensed user name and organization.

It is recommended that these status functions can be used to provide information to the application end-user in the status bar or "About" box as well as allowing applications using the C3Dserver to require that a current version is loaded.

C3Dserver Status

This function returns the current software license status (registered or unregistered) of the C3Dserver application. The software license registration is separate from the operating system registration – the C3Dserver dynamic link library must be registered with the operating system in order to function correctly and is normally registered with the operating system by the standard Motion Lab Systems installation application.

An unlicensed copy of the C3Dserver can be run on any system but it will run slowly as the unlicensed version includes a delay during each C3Dserver function – this delay does not exist in the licensed version which will perform mach faster. Licensed versions of the C3Dserver for individual use or commercial redistribution can be purchased from Motion Lab Systems for those users requiring greater performance or for commercial product development and distribution.

Typically, this function would be used to obtain the registration status for debugging or display in an application. It is recommended that all programs that use the C3Dserver should check the registration status as soon as the application loads and before the main initialization sub-routines and display the status within the application – the supplied example applications all display the C3Dserver status and version number in their "Help About" dialog box.

HRESULT GetRegistrationMode (int *pMode)

Return Value

A standard **HRESULT** value.

Parameters

pMode - [out] This is the return value, which is the registration mode of the C3D Server module. Possible values are:

- 0 – Unregistered (versions prior to 1.129 only).
- 1 – Unregistered - evaluation mode (slow execution).
- 2 – Registered – the C3Dserver will operate at full speed.

Remarks

Both the Registered and Unregistered versions of the C3Dserver implement an identical set of functions – the only difference between the two versions is speed of execution.

Starting with Version 1.130, the C3Dserver can be installed without requiring that its registration mode is set – effectively the “unregistered” and “evaluation” modes are now identical. This means that the C3Dserver DLL can be easily supplied with non-commercial applications as it can be installed on any system simply by calling regsvr32 and creating the appropriate registry entries.

Example

```
Dim nRet As Integer
Dim sReg As String
nRet = itf.GetRegistrationMode
sReg = Switch(nRet = 0, "Unregistered", nReg = 1,
"Unregistered", nReg = 2, "Registered")
```

```
The function returns the current registration status
of the C3Dserver.
```

C3Dserver - User Name

This function returns the name of the registered C3Dserver user. The standard version of the C3Dserver will always return the string "Evaluation Copy".

Licensed versions of the C3Dserver will return a value determined by Motion Lab Systems that identifies the licensed user. Typically this information would be used in the "Help: About" display or displayed when the application starts.

HRESULT GetRegUserName (BSTR *pName)

Return Value

A standard **HRESULT** value.

Parameters

pName - [out] This is the return value, which is the name of the user to whom the copy of the C3D Server is registered.

Example

```
Dim sName As String

sName = itf.GetRegUserName

   The variable sName holds the name of the person to
   whom the server has been registered.
```

C3Dserver - Organization

This function returns the name of the organization that distributed this version of the C3Dserver. The standard version of the C3Dserver will always return the string "Motion Lab Systems, Inc."

Licensed versions of the C3Dserver will return a value determined by Motion Lab Systems that identifies the licensed organization. Typically this information would be used in the "Help: About" display or displayed when the application starts.

HRESULT GetRegUserOrganization (BSTR *pOrg)

Return Value

A standard **HRESULT** value.

Parameters

pOrg - [out] This is the return value, which is the organization of the user to whom the copy of the C3D Server is registered.

Example

```
Dim sOrg As String
sOrg = itf.GetRegUserOrganization

   The variable sOrg holds the name of the organization
   that distributed this version of the server.
```

C3Dserver Version

This function returns the version number of the registered C3Dserver. Any application written using the C3Dserver should provide the user with a means of displaying the C3Dserver version number. This can be used internally and for diagnostic purposes to determine if a recently added feature is available.

HRESULT GetVersion (BSTR *pVersion)

Return Value

A standard **HRESULT** value.

Parameters

pVersion - [out] This is the return value which is the version of the C3D Server module.

Example

```
Dim sVersion As String
sVersion = itf.GetVersion
```

```
The variable sVersion holds the version of the server
module.
```

C3D File Functions

The C3Dserver provides a number of functions to create, open, close and save C3D files, determine if an open C3D file has been modified (and therefore should be saved) as well as a function to create a new C3D file. The function to create a new C3D file allows the user to create a basic C3D file, complete with parameters, very quickly by specifying the basic required parameter values automatically when the file is first created.

Open a C3D file

This function opens a C3D file in one of three modes documented below. Note that a C3D file must be opened in Mode 3 before any editing functions can be called. The file image on disk of C3D file that has been opened will not be changed until the C3D file has been saved thus any changes to the C3D file may be saved to a different file name or simply canceled at any time before closing the C3D file.

Note that this also means that if your application exits unexpectedly then all modifications to the C3D file will be lost unless you take steps to save changes and edits as they are made.

HRESULT Open (BSTR sFilename, int nMode, int * pReturn)

Return Value

A standard **HRESULT** value.

Parameters

sFilename - [in] The name of the file to be opened.

nMode - [in] The mode in which the C3D file should be opened. This can be used to control just what amount of the data in the file is read. The valid values are

- 1 – Only the header of the file is read – this allows an application to open and quickly read the file header providing basic information about the file contents.
- 2 – The header and the group and parameter information is read. This is not as fast as reading the file header but provides detailed information about the file contents without actually reading the file data.

- 3 – The entire contents of the file, header, parameters and data, is read in one operation.

  If you use a value other than the ones mentioned above, all of the data in the file will be read.

  If you plan to perform any editing at all on the C3D file, then it has to be opened in mode 3.

pReturn - [out] A pointer to an integer. The value returned here is zero if the file is opened successfully or one if it could not be opened.

Remarks

When a C3D file is opened, its contents, depending on the mode specified, are read into memory and stored. You must call **Close** for each file that you have opened.

Example

```
sFilename = "C:\test.c3d"
nReturn = itf.Open(sFilename, 3)
```

```
This opens the C3D file whose name is in the variable
sFilename in such a way that all of the data in the
file is accessible.  If nReturn is 0, it implies that
the file was opened correctly.
```

Determine if open C3D file was modified

When a C3D file has been opened the entire contents of the file is stored in memory. Use this function to determine if any part of an open C3D file has been changed before it is closed.

HRESULT IsModified (BYTE *byModified)

Return Value

A standard **HRESULT** value.

Parameters

byModified - [out] Returns 1 if the file has been modified or 0 if the file has not been modified.

Remarks

All the data that has been read using the Open function is released when you call this function. This function must be called if you have a opened a file using the **Open** function.

Example

```
Dim nMod As Integer
nMod = itf.IsModified
```

```
   nMod contains 1 if the file has been modified using
   the server since you opened the file, 0 indicates
   that you have not modified it.
```

Save C3D file

Save the contents of an open C3D file to disk using either the original data format or selecting a new data format (INTEL (MS-DOS), DEC, SGI etc). Note that the data type of the original C3D file will always be preserved – floating-point data C3D files will always be written as floating-point data while integer C3D files will always be written as integer. This avoids the possibility of losing data precision when data is converted from floating-point format to integer format.

HRESULT SaveFile (BSTR sFilename, int nFileType, int * pReturn)

Return Value

A standard **HRESULT** value.

Parameters

sFilename - [in] This is the name of the file to which the data will be saved. If this is empty, the data will be saved to the open file.

nFileType - [in] If this parameter is –1 the data is saved to the existing file type. If you want to save to a different file type, use the following values:

- Intel (MS-DOS) Format : 1
- DEC Format : 2
- SGI Format : 3

pReturn - [out] A value of 1 is returned if the file is saved. If the file is not saved a value of zero is returned.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2000 – If the input or output file could not be opened.

Example

```
nRet = itf.SaveFile("", -1)

   This saves the open file to the same name and the
   same data type

sFile = "somenewname.c3d"
nRet = itf.SaveFile(sFile, 3)

   This saves the open file to the file somenewname.c3d
   having data in SGI Format.

   In either case, nRet contains 1, if the file was
   saved correctly.
```

Close a C3D file

This function closes a C3D file that has been previously opened and releases the memory. This function does not automatically save the C3D file.

HRESULT Close()

Return Value

A standard **HRESULT** value.

Parameters

None.

Remarks

All the data that has been read using the Open function is released when you call this function. This function must be called if you have opened a file using the **Open** function.

Example

```
itf.Close
   This call closes the open C3D file.
```

Create a new C3D file

This function creates a new C3D file in memory using the supplied parameters. The created C3D file is not saved to disk using this function. You have to call SaveFile later if you want this new file to be written to disk.

HRESULT Open (BSTR sFile, int nFileType, int nDataType, int nAnalog, int nAVRatio, int nMarkers, float fRate, float fScale, int nFrames, int *pReturn)

Return Value

A standard **HRESULT** value.

Parameters

sFile - [in] The name to be given to the C3D file. Since the file is not going to be saved to disk, this parameter can be empty

nFileType - [in] The type of file you want to create. Valid values are:

- Intel (MS-DOS) Format : 1
- DEC Format : 2
- SGI Format : 3

If you enter a value other than these, the module will use the Intel format.

nDataType - [in] The format in which analog and point data will be stored. Valid values are:

- Integer : 1
- Floating-point : 2

The default format is the Integer format, which will be used if you enter an incorrect value.

nAnalog - [in] The number of analog channels to be created in the file. This cannot be less than zero.

nAVRatio - [in] The number of analog frames per video frame. This cannot be less than zero. If this value is zero, the number of analog channels is set to zero.

nMarkers - [in] The number of markers to be created. This value should not be less than zero.

fRate - [in] The sampling rate of the video frames. This should be greater than zero. If this is not the case, the application will throw an error **2046** and the function will fail.

fScale - [in] The scaling factor of point data. This should be greater than zero. If it is not, the application with convert it to a positive value.

nFrames - [in] The number of frames to be created. This should be greater than or equal to zero. If not the application will throw an error **2038** and the function will fail.

pReturn - [out] A pointer to an integer. The value returned here is one if the file is created successfully or zero if it could not be created.

Remarks

When a file is created, its contents are in memory. You must call **Close** for each file that you have created in order to create and save the file to disk. The server also creates the following parameters:

- ANALOG:GEN_SCALE – The general scaling factor. Its value is set to one by default – you must set the correct value for your application.
- ANALOG:RATE – The sampling rate of analog data. This is the product of fRate and nAVRatio.
- ANALOG:USED - The number of analog channels created.
- ANALOG:SCALE – The scale factor for each channel. They are all set to one by default – you must set the correct value for your application.
- ANALOG:OFFSET – The offset value for each channel. They are all set at 2048 by default – you must set the correct value for your application.
- ANALOG:UNITS – The units of measurement. They are all empty – you should set the correct values.
- ANALOG: LABELS – The labels for each channel. It is of the form CHN, where N is the channel number starting with one, – you will probably want to enter values that indicate the channel contents.
- ANALOG:DESCRIPTIONS – The description for each channel. It is of the form Channel N, where N is the channel number starting with 1 – you will probably want to enter values that indicate the channel contents.
- POINT:RATE – The sampling rate of video data. This is fRate.
- POINT:USED - The number of point markers created.
- POINT:SCALE – The scale factor for point data. This is the fScale value.
- POINT:DATA_START – The starting record at which the analog and point data can be found.
- POINT:FRAMES – The number of frames of video data. This is the nFrames value.
- POINT: LABELS – The labels for each marker. It is of the form MKN, where N is the marker number starting with one.
- POINT:DESCRIPTIONS – The description for each marker. It is of the form Marker N, where N is the marker number starting with one.
- FORCE_PLATFORM:USED – The number of force plates used. This is set to zero.

Example

```
Dim sFile As String
Dim nReturn As Integer
sFile = "c:\newfile.c3d"
```

```
nReturn = itf.NewFile(sFile, 1, 1, 25, 10, 20, 60, 0.15,
200)
```

This creates a C3D file with name sFile. It is opened with the following characteristics:

- Integer File Type
- Integer Data Type
- 25 Analog Channels
- 10 analog channels per video channel
- 20 markers
- Video Sampling frequency of 60 Hz.
- Scaling factor of point data of 0.15
- 200 video frames

If the function returns 1, it implies that the file has been created.

C3D Header Functions

The C3D file header provides basic information about the C3D file and its contents. Much of this information is a copy of data held in the C3D file Parameters and is provided in the header so that basic information about the C3D file can be obtained by simply by reading the file header. As a result, most of the C3Dserver functions for the header simply read information from the C3D file header as the information stored there in most cases is simply a copy of information stored in the Parameter Block and is updated by the C3Dserver when the associated parameters are written.

The most important information within the C3D file header stores the location of the start of the 3D and analog data block as well as the type of C3D file format.

Note that a small number of time-based events may be stored within the C3D file header.

Get 3D markers used

This function returns the C3D file header record of the number of 3D markers stored in the C3D file. This value is also generally available as the parameter POINT:USED – both data items should always return the same value in a correctly formatted C3D file. This value is set by writing to the parameter POINT:USED – the header value can not be set independently of the parameter value.

HRESULT GetNumber3DPoints (int * *n3dPoints*)

Return Value

A standard **HRESULT** value.

Parameters

n3dPoints - [out] A pointer to an integer. The value returned here is the number of markers whose data has been collected in the file. This value is taken from the header of the C3D file.

Example

```
nPoints = itf.GetNumber3DPoints
```

```
   This gives you the number of markers for which data
   is present in the file.  This information is taken
   from the header record of the file.
```

Get analog channels used

Returns the C3D file header record of the number of analog channels stored in the C3D file. This value is also generally available as the parameter ANALOG:USED. The parameter and header values should be identical. This value is set by writing to the parameter ANALOG:USED – the header value can not be set independently of the parameter value.

HRESULT GetAnalogChannels (int * *nAnalog*)

Return Value

A standard HRESULT value.

Parameters

nAnalog - [out] A pointer to an integer. The value returned here is the number of analog channels whose data has been collected in the file. This value is taken from the header of the C3D file.

Example

```
nAnalog = itf.GetAnalogChannels

   This gives you the number of analog channels for
   which data is present in the file.  This information
   is taken from the header record of the file.
```

Get 3D frame range from header

This function allows you to read either the first or last 3D frame numbers from an open C3D file and will always return the values stored in C3D header.

HRESULT GetVideoFrameHeader (int nStartEnd, int * nFrame)

Return Value

A standard HRESULT value.

Parameters

nStartEnd - [in] This is an integer value which tells the function if you want to retrieve the start frame or the end frame. A value of 0 gives you the start frame while a value of 1 gives you the end frame.

nFrame - [out] A pointer to an integer. The value returned here is value of the start or end frame of point data that has been collected. This value is the one found in the header of the C3D file.

Remarks

The value returned here is from the header of the C3D file. However, in some C3D files this may not reflect the true number of frames of data. It is safer to use the GetVideoFrame function if you want to get the correct start and end frame.

Example

```
nStartFrame = itf.GetVideoFrameHeader(0)
nEndFrame = itf.GetVideoFrameHeader(1)
```

```
This gives you the first frame of video data in the
nStartFrame variable and the last frame in the
nEndFrame variable.  This information is taken from
the header record of the file.
```

Get interpolation gap

Returns the interpolation gap value stored in the C3D file header. This value does not have any corresponding parameter value.

HRESULT GetMaxInterpolationGap (int * *nIntGap*)

Return Value

A standard HRESULT value.

Parameters

nIntGap - [out] A pointer to an integer. The value returned here is the maximum interpolation gap in fields taken from the header of the C3D file.

Example

```
nGap = itf.GetMaxInterpolationGap

   This gives you maximum interpolation gap.  This
   information is taken from the header record of the
   file.
```

Set interpolation gap

This function allows you to set the interpolation gap value stored in the C3D file header. This header value does not have a corresponding parameter stored in the Parameter Block.

HRESULT SetMaxInterpolationGap (int *nIntGap*, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIntGap - [in] A pointer to an integer. This is the new value of the interpolation gap. This value must be greater than or equal to zero.

pReturn - [out] A pointer to an integer. The value returned here is 1 if the function was successful or 0 if it was not.

Example

```
nReturn = itf.SetMaxInterpolationGap(5)

   This function changes the value of the maximum
   interpolation gap.  In the above example, this value
   has been changed to 5.
```

Get 3D scale factor

Returns the value of the 3D scale factor stored in the C3D file header. This value may also be read from the parameter POINT:SCALE and should return the same value as the header. This value is set by writing to the parameter POINT:SCALE – the header value can not be set independently of the parameter value.

HRESULT GetHeaderScaleFactor (float * *fScale*)

Return Value

A standard HRESULT value.

Parameters

fScale - [out] A pointer to an floating-point number with single precision. This returns the scaling factor value for 3D data from the C3D file header.

Remarks

The header SCALE value can be used to determine if the 3D and analog data records are stored in integer or floating-point format. The 3D scale factor is always negative if the C3D file uses a floating-point data format.

Example

```
fScale = itf.GetHeaderScaleFactor

   This gives you the value of the scale factor found in
   the header of the file.  This value will be positive
   if the data is stored in integer format and will be
   negative if the data is stored in Floating-point
   format.
```

Get start of data

This function returns the value of the record where the 3D and analog data starts based on the information stored in the C3D file header.

HRESULT GetStartingRecord (int * *nRecord*)

Return Value

A standard HRESULT value.

Parameters

nRecord - [out] A pointer to an integer. The value returned here is value of the record where the actual data starts.

Remarks

This value is also stored as the parameter POINT:DATA_START that should always contain the same value as stored in the C3D file header.

Example

```
nStartRecord = itf.GetStartingRecord

   This returns the record number at which the analog
   and video data are stored.
```

Get analog to video ratio

Returns the number of analog frames stored for each video frame in the C3D file based on information in the C3D file header.

HRESULT GetAnalogVideoRatio (int * *nAVRatio*)

Return Value

A standard HRESULT value.

Parameters

nAVRatio - [out] A pointer to an integer. The value returned here is the number of analog frames collected per video field.

Remarks

This information is also available in the parameter ANALOG:RATE that should contain an identical value to the C3D file header.

Example

```
nRatio = itf.GetAnalogVideoRatio

   This is the value from the header of the file that
   indicates the number of analog frames per frame of
   video data.
```

Get video frame rate

Returns the 3D point sample rate in Hertz as read from the C3D file header.

HRESULT GetVideoFrameRate (int * fVideoRate)

Return Value

A standard HRESULT value.

Parameters

fVideoRate - [out] A pointer to an single precision floating-point number. The value returned here is value of the video rate in Hz from the header of the C3D file.

Remarks

This value is also available as the parameter POINT:RATE.

Example

```
fRate = itf.GetVideoFrameRate

   This gives you the rate in Hertz at which the video
   data is collected in the file.  This information is
   taken from the header record of the file.
```

Get C3D file type

Returns the C3D file floating point data storage type.

HRESULT GetFileType (int * pType)

Return Value

A standard HRESULT value.

Parameters

pType - [out] A pointer to an integer. This shows you the type of file. The values can be:

- Intel (MS-DOS) Format : 1
- DEC Format : 2
- SGI Format : 3

Example

```
nType = itf.GetFileType

   This function returns in the format in which numeric
   values are stored in the C3D file.  The values can be
   1, 2 or 3, which imply Intel, DEC, and SGI
   respectively.  This information is taken from the
   header record of the file.
```

Get data type

C3D files can store 3D and analog data using either floating point or integer forms. This function reports which type is used in the open C3D file.

HRESULT GetDataType (int * *pType*)

Return Value

A standard HRESULT value.

Parameters

pType - [out] A pointer to an integer. This shows you format in which analog and point data are stored. The values can be:

- Integer : 1
- Floating-point : 2

Example

```
nDataType = itf.GetDataType
```

```
   This indicates whether the manner in which the analog
   and video data have been stored in the file.  This
   can be Integer (1) or Floating-point (2).  This
   information is taken from the header record of the
   file.
```

Get header event count

This function returns the number of events stored in the C3D file header.

HRESULT GetNumberEvents (int * *pEvents*)

Return Value

A standard HRESULT value.

Parameters

pEvents - [out] A pointer to an integer. The value returned here is the number of events in the header of the C3D file.

Example

```
nEvents = itf.GetNumberEvents

   This function returns the number of events that have
   been stored in the header of the file.
```

Get event time

This function returns the time for the specified event in the C3D file header.

HRESULT GetEventTime (int *nIndex*, float * *fTime*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This parameter is the index of the event whose name you want to find. This is a zero based number and thus must be in the range of 0 to (GetNumberEvents-1).

fTime - [out] A pointer to an floating-point number. The value returned here is time at which the event you have pointed to occurred. If *nIndex* is invalid, 0 is returned and an error is reported.

Example

```
fTime = itf.GetEventTime(0)
```

```
   This function returns the time associated with an
   event.  In the example, we get the time associated
   with the first event in the file header.
```

Get event label

This function returns the label for the specified event in the C3D file header.

HRESULT GetEventLabel (int *nIndex*, BSTR * *sLabel*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This parameter is the index of the event whose name you want to find. This is a zero based number and thus must be in the range 0 to (GetNumberEvents-1).

sLabel - [out] A pointer to a string. The value returned is the label assigned to the event. If *nIndex* is invalid, 0 is returned and an error is reported.

Example

```
sLabel = itf.GetEventLabel(0)
```

This function returns the four character label associated with an event. In the example, we get the label associated with the first event in the file header.

Get event status

This function returns the status (on or off) of the specified event in the C3D file header.

HRESULT GetEventStatus (int nIndex, BYTE * byStatus)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the event whose name you want to find. This is a zero based number and thus must be in the range 0 to (GetNumberEvents-1).

byStatus - [out] A pointer to byte. This returns the status as 0 or 1. 0 implies ON while 1 implies OFF.

Example

```
byStatus = itf.GetEventStatus(0)

   This function returns the status of the event.  In
   the example, we get the status of the first event in
   the file header.  0 implies the event is ON while 1
   implies that it is OFF.
```

Create event

This function creates the specified event in the C3D file header.

HRESULT AddEvent (BSTR sLabel, BYTE byStatus, float fTime, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

sLabel - [in] This is the 4-character label that the event can have.

byStatus - [in] This is the status of the event – 0 indicates ON while 1 indicates OFF

fTime - [in] This is a floating-point value that indicates the time of the event.

pReturn - [out] The index of the event that was added. –1 if the event could not be added.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2016 – If the number of events exceeds the maximum number allowed (18)

Example

```
sLabel = "TO1"
byStatus = 0
fTime = 1.21

nEvent = itf.AddEvent(sLabel, byStatus, fTime)

   This call will add an event labeled TO1 at time 1.21
   seconds with an ON status.  nEvent will have the
   index of the added event.
```

Modify event label

This function modifies the specified event label stored in the C3D file header.

HRESULT SetEventLabel (int nIndex, BSTR sLabel, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the event whose label you want to change. This is a zero based number and thus must be in the range 0 to (GetNumberEvents-1).

sLabel - [in] This is the new label for the event. This must not exceed 4 characters.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
sNewLabel = "TO2"
nEvent = 0 ‘ the event to be modified
nSuccess = itf.SetEventLabel(nEvent, sNewLabel)
```

This call will change the label of the first event to TO2.

Modify event time

This function changes the time value of the specifed event stored in the C3D file header.

HRESULT SetEventTime (int nIndex, float fTime, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the event whose time you want to change. This is a zero based number and thus must be in the range 0 to (GetNumberEvents-1).

fTime - [in] This is the new time for the event.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
fTime = 1.6
nEvent = 0 ‘ the event to be modified
nSuccess = itf.SetEventTime(nEvent, fTime)
```

This call will change the event time of the first event to 1.6 seconds.

Modify event status

This function sets the status of the specified event stored in the C3D file header.

HRESULT SetEventStatus (int nIndex, BYTE byStatus, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the event. This is a zero based number and thus must be in the range 0 to (GetNumberEvents-1).

fTime - [in] This is the status of the event. 0 implies ON while 1 implies OFF.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
byStatus = 1
nEvent = 0 ‘ the event to be modified
nSuccess = itf.SetEventStatus(nEvent, byStatus)
   This call with turn off the first event
```

Delete event

This function removes a previously stored event from the C3D file header.

HRESULT DeleteEvent (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the specified event. This is a zero based number and thus must be in the range 0 to (GetNumberEvents-1).

pReturn - [out] 1 if the event was deleted, 0 if it was not.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nEvent = 2 ‘ the event to be deleted
nSuccess = itf.DeleteEvent(nEvent)
   This call will delete the third event.
```

C3D Parameter Block

The C3D parameter block contains information about the data contained in the C3D file. This data is referred to as *Parameters* and is organized into *Groups* of similar *Parameters* to provide a basic level of organization. The C3Dserver supports a number of functions that operate on the Parameter Block as a whole.

Strict Parameter Checking

The C3D format specifies that the first six characters of the parameter/group name have to be unique. This requirement is enforced by default in the C3Dserver DLL. However, if you need to use parameters that cannot pass this test, then you can use this function to turn off strict checking. The checking of Group/Parameter names are used in the following functions: AddGroup, SetGroupName, AddParameter and SetParameterName.

HRESULT SetStrictParameterChecking(int nValue)

Return Value

A standard HRESULT value.

Parameters

nValue- [in] A pointer to an integer. If 0, then the strict parameter checking is not used. If you pass any other value, then the uniqueness of the first six characters is enforced.

Remarks

Added in Version 1.125 of the C3Dserver, this function allows users to work with C3D files that do not strictly adhere to the C3D specification for group and parameter names. Use this function with care, as C3D files that contain such group or parameter names may not be readable in some software applications.

Example

```
itf.SetStrictParameterChecking(0)
```

This function call turns off strict checking of parameter names. You could now have a parameter named ACTUAL_START_FIELD and create a new one called

ACTUAL_END_FIELD and the parameter creation function would not return an error.

```
itf.SetStrictParameterChecking(1)
```

This function call turns on strict checking of parameter names. This ensures that the first six characters of group and parameter names are unique.

Compress parameter block

This function ensures that any empty parameter blocks are removed when the C3D file is saved. This is useful if you have deleted a large number of parameters from the file.

HRESULT CompressParameterBlocks (int Compress)

Return Value

A standard HRESULT value.

Parameters

Compress - [in] - Pass a value of 0 (zero) if you do not want parameter blocks to be removed. Pass any positive value if you want them to be removed.

Remarks

By default the server does not remove parameter blocks. Please call this function before you call the Save function to ensure that empty parameter blocks are removed.

Example

To remove parameter blocks :

```
Itf. CompressParameterBlocks(1)
```

To **not** remove parameter blocks :

```
Itf. CompressParameterBlocks(0)
```

C3D Group functions

Groups are the basic level of organization of information within the Parameter Block. The C3Dserver provides a number of functions that allow the user complete control over the parameter Groups allowing them to be created, deleted and edited at will.

Get group count

This function returns the number of parameter groups stored in the C3D file parameter section.

HRESULT GetNumberGroups (int * *nGroups*)

Return Value

A standard HRESULT value.

Parameters

nGroups - [out] A pointer to an integer. The value returned here is the number of groups in the C3D file.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nGroups = itf.GetNumberGroups()

   nGroups will contain the number of groups present in
   the file.
```

Get group index

This function returns the internal C3Dserver index value for a specified group.

HRESULT GetGroupIndex (BSTR *sGroup*, int * *pIndex*)

Return Value

A standard HRESULT value.

Parameters

sGroup - [in] This is the name of the Group whose index you want to find. This value is case-insensitive and all leading and trailing spaces will be removed.

nRecord - [out] A pointer to an integer. The value returned is the index of the Group with the given name. You can use this index as a parameter with other functions. You must however remember that if you were to delete a group, then this index might no longer be valid or could point to some other group. Thus, it is preferable that you retrieve this index as and when needed. This value is set to -1, if there is no group with the given name.

Remarks

When the C3Dserver reads the parameter section of a file, it stores the group data in an array starting from 0 and going to NumberGroups-1. The group index is used by the C3Dserver as an internal index number for references to each specific group. Thus, the group number (assigned to each group in the C3D) can be preserved independently.

So using this function you get the internal index of the group you want and then for all future group based calls you can use this index. Of course, if you delete a group, you will have to get this index again because it might have changed.

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)

   nIndex will contain the index of the ANALOG group in
   the file.  This index can then be used to access
   other Group related functions like GetGroupName
```

Get group name

This function returns the name of the group specified by the specified index value.

HRESULT GetGroupName (int *nIndex*, BSTR * *pName*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group whose name you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfGroups-1).

pName - [out] A pointer to an string. The value returned here is the name of the group at the given index. If *nIndex* is invalid, an empty string is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
sName = itf.GetGroupName(2)

   sName will contain the name of the group that has
   index number 2.
```

Get group number

This function returns a unique identifying number stored in the parameter section of the C3D file that is used to reference each specific group.

HRESULT GetGroupNumber (int *nIndex*, int * *pNumber*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group whose number you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfGroups-1).

pNumber - [out] A pointer to an integer. The value returned here is the number of the group at the given index. If *nIndex* is invalid, zero is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)
nNumber = itf.GetGroupNumber(nIndex)
   This set of function calls will give you the Group
   Number of the ANALOG group.
```

Get group status

This returns the status of the specified group (locked or unlocked).

HRESULT GetGroupLock (int *nIndex*, BYTE * *byLock*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group whose number you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfGroups-1).

byLock - [out] A pointer to a byte. The value returned is 1 if the group has been locked or 0 if the group has not been locked.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)
byLock = itf.GetGroupLock(nIndex)
   This set of function calls will tell you whether the
   ANALOG group is Locked.
```

Get group description

This returns the description string (if any) of the specified group.

HRESULT GetGroupDescription (int nIndex, BSTR * pDesc)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group whose description you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfGroups-1).

pDesc - [out] A pointer to a string. The value returned here is the description of the group at the given index. If *nIndex* is invalid, an empty string is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetGroupIndex(“POINT”)
sName = itf.GetGroupDescription(nIndex)
   This set of function calls will give you the
   description of the POINT group.
```

Set group name

This sets the name for the specified group.

HRESULT SetGroupName (int nIndex, BSTR sName, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group. This is a zero based number and thus must be in the range 0 to (GetNumberGroups-1).

sName - [in] This is the new name for the group. This name should not belong to any other group.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

By default the C3Dserver will check that the first six characters of a new group name are unique unless StrictParameterChecking has been disabled. Attempts to rename a group to a name that contain the same six first six characters as an existing group will fail unless StrictParameterChecking has been disabled.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2020 – The group name is empty.

2022 – The group name already exists.

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)

nSuccess = itf.SetGroupName(nIndex, “ANALOG1”)

   This set of function calls will modify the name of
   the ANALOG group to ANALOG1.
```

Set group number

This sets the number for the specified group.

HRESULT SetGroupNumber (int nIndex, int nNumber, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group. This is a zero based number and thus must be in the range 0 to (GetNumberGroups-1).

nNumber - [in] This is the new number for the group. This number should not belong to any other group and should be a number less than zero.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2019 – The group number is greater than or equal to zero.

2021 – The group number already exists.

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)
nSuccess = itf.SetGroupNumber(nIndex, -25)
   This set of function calls will change the Group
   Number of the ANALOG group to -25.
```

Set group description

This sets the description string for the specified group.

HRESULT SetGroupDescription (int nIndex, BSTR sDesc, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group. This is a zero based number and thus must be in the range 0 to (GetNumberGroups-1).

sDesc - [in] This is the description for the group.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)
sDesc = “Analog Parameters”
nSuccess = itf.SetGroupDescription(nIndex, sDesc)
   This set of function calls will change the
   description associated with the ANALOG group.
```

Set group status

This sets the status of the specified group (locked or unlocked)

HRESULT SetGroupLock (int nIndex, BYTE byLock, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group. This is a zero based number and thus must be in the range 0 to (GetNumberGroups-1).

byLock - [in] This is the lock status of the group. 1 indicates that the group should be locked, while 0 indicates that it is unlocked.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)
nSuccess = itf.SetGroupStatus(nIndex, 1)
   This set of function calls will Lock the ANALOG
   group.
```

Create group

This creates a group parameter record.

HRESULT AddGroup (int nNumber, BSTR sName, BSTR sDesc, BYTE byLock, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nNumber - [in] This is the number for the group. If you use a value of 0, a unique group number will be found. If you use a value less than zero, it should be a value that does not belong to any other groups.

sName - [in] The name of the group. This should not be empty and should not belong to any other group.

sDesc - [in] The description for the group.

byLock - [in] The lock status of the group.

pReturn - [out] -1 if the group could not be added. The index of the group if it was added successfully.

Remarks

By default the C3Dserver will check that the first six characters of the new group name are unique unless StrictParameterChecking has been disabled. Attempts to create group names that contain the same six first six characters as an existing group will fail unless StrictParameterChecking has been disabled.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2019 – The group number is greater than zero.

2021 – The group number already exists.

2020 – The group name is empty.

2022 – The group name already exists.

Example

```
nNumber = -15
sName = “TEST”
sDesc = “Test Group”
byLock = 0
nIndex = itf.AddGroup(nNumber, sName, sDesc, byLock)
   This function will create a group named TEST.
```

Delete group

This deletes the specified parameter group and all parameter values associated with the specified group.

HRESULT DeleteGroup (int nIndex, BYTE byDeleteParams, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the group. This is a zero based number and thus must be in the range 0 to (GetNumberGroups-1).

byDeleteParams - [in] If this parameter is set to a non-zero value, all the parameters belonging to the group will also be deleted.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetGroupIndex(“ANALOG”)

nSuccess = itf.DeleteGroup(nIndex, 1)

   This set of function calls will delete the ANALOG
   group and all the parameters that belong to this
   group.
```

C3D Parameter functions

Information about the 3D and analog data within the C3D file is stored in Parameters together with information about the structure of the C3D file itself. The C3Dserver provides numerous functions that allow the user complete control over the C3D file parameters allowing them to be created or deleted, and their contents edited at will.

Get parameter count

This returns the total number of parameters in the open C3D file.

HRESULT GetNumberParameters (int * *nParams*)

Return Value

A standard HRESULT value.

Parameters

nParams - [out] A pointer to an integer. The value returned here is the number of parameters in the C3D file.

Remarks

When searching for a specific parameter it is useful to know how many parameters exist within the C3D parameter section in order to determine if the search has completed without finding the specific parameter name.

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nParams = itf.GetNumberParameters
   nParams will contain the number of parameters that
   are present in the file.
```

Get parameter index

This function returns the internal C3Dserver index number for the specified group parameter.

HRESULT GetParameterIndex (BSTR *sGroup*, BSTR *sParam*, int * *pIndex*)

Return Value

A standard HRESULT value.

Parameters

sGroup - [in] This is the name of the Group to which the parameter you want to find belongs to. This value is case-insensitive and all leading and trailing spaces will be removed.

sParam - [in] This is the name of the Parameter whose index you want to find. This value is case-insensitive and all leading and trailing spaces will be removed.

nRecord - [out] A pointer to an integer. The value returned is the index of the Parameter with the given name belonging to the given Group. You can use this index as a parameter to other functions. You must however remember that if you were to delete a parameter or group, then this index might no longer be valid or could point to some other parameter. Thus, it is preferable that you retrieve this index as and when needed. This value is set to -1, if there is no parameter with the given name.

Remarks

When the C3Dserver reads the parameter data, it is stored internally in an array starting from 0 and going to NumberParameters-1. The ParameterIndex is a unique number that refers to a specific parameter within a group. You can use this function you get the internal index of the parameter that you want and then all future parameter based calls can use this index. Of course, if you delete a parameter, you will have to get this index again because it might have changed.

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex("ANALOG", "LABELS")

   This will give you the index of the ANALOG:LABELS
   parameter.  You can then use this index for all
   parameter related calls.
```

Get parameter name

This returns the name of the parameter with the specific index value.

HRESULT GetParameterName (int *nIndex*, BSTR * *pName*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter whose name you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

pName - [out] A pointer to an string. The value returned here is the name of the parameter at the given index. If *nIndex* is invalid, an empty string is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
sName = itf.GetParameterName(10)

   This will give you the name of the eleventh
   parameter.
```

Get parameter number

This returns the internal reference number used by a particular parameter within the C3D parameter section.

HRESULT GetParameterNumber (int *nIndex*, int * *pNumber*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter whose number you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

pNumber - [out] A pointer to an integer. The value returned here is the number of the Parameter at the given index. If *nIndex* is invalid, zero is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

nNumber = itf.GetParameterNumber(nIndex)

   This set of calls will give you the Parameter Number
   of the ANALOG:LABELS parameter.
```

Get parameter status

This returns the status (locked or unlocked) for the specified parameter.

HRESULT GetParameterLock (int *nIndex*, BYTE * *byLock*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter whose number you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

byLock - [out] A pointer to a byte. The value returned is 1 if the parameter has been locked or 0 if the group has not been locked.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
byStatus = itf.GetParameterStatus(nIndex)
   This set of calls will tell you if the ANALOG:LABELS
   parameter is locked.
```

Get parameter description

This returns the description string for the specified parameter.

HRESULT GetParameterDescription (int *nIndex*, BSTR * *pDesc*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the Parameter whose description you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

pDesc - [out] A pointer to a string. The value returned here is the description of the parameter at the given index. If *nIndex* is invalid, an empty string is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
sDesc = itf.GetParameterDescription(nIndex)
   This set of calls will give you the description of
   the ANALOG:LABELS parameter.
```

Get parameter type

This returns the storage method (or type) for the specified parameter.

HRESULT GetParameterType (int *nIndex*, int * *pType*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the Parameter whose data type you want to find. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1). If *nIndex* is invalid, zero is returned and an error is reported.

pType - [out] A pointer to an integer. The value returned here indicates the type of data that is stored in the parameter. The different values are:

- Character : -1
- Byte : 1
- Integer : 2
- Floating-point : 4

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nType = itf.GetParameterType(nIndex)
   This set of calls will give you the type of data that
   is stored in the ANALOG:LABELS parameter.
```

Get parameter dimensions

This returns the number of dimensions for a given parameter.

HRESULT GetParameterNumberDim (int *nIndex*, int * *pNumDim*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the Parameter. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

pNumDim - [out] A pointer to an integer. The value returned here is the number of dimensions. If *nIndex* is invalid, -1 is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nNumDim = itf.GetParameterNumberDim(nIndex)
   This set of calls will give you the number of
   dimension in the ANALOG:LABELS parameter.
```

Get parameter dimension

This returns the size of the specified parameter dimension.

HRESULT GetParameterDimension (int *nIndex*, int nDim, int * *pValue*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the Parameter. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

nDim - [in] This is the index of the dimension with the dimension. This is a zero based number and thus must be in the range 0 to one less than the number of dimensions returned by the GetParameterNumberDim function.

pValue - [out] A pointer to an integer. The value returned here is the value of the dimension asked for. If *nIndex* or *nDim* are invalid, -1 is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nNumDim = itf.GetParameterNumberDim(nIndex)
For I = 0 To nNumDim-1
      NDimension = itf.GetParameterDimension(nIndex, I)
Next I
   This set of calls will allow you to enumerate the
   dimensions of the ANALOG:LABELS parameter.
```

Get parameter length

HRESULT GetParameterLength (int *nIndex*, int * *pLength*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the Parameter. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1).

pLength - [out] A pointer to an integer. The value returned here is the number of different items stored in this parameter. If *nIndex* is invalid, -1 is returned and an error is reported.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nItems = itf.GetParameterLength(nIndex)
   This set of calls will allow you to get the number of
   items that are stored in the ANALOG:LABELS parameter.
```

Get parameter value

This function returns the value of the specified parameter.

HRESULT GetParameterValue (int *nIndex*, int *nItem*, VARIANT * *pValue*)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the Parameter. This is a zero based number and thus must be in the range 0 to (NumberOfParameters-1). If *nIndex* is invalid, an error is reported.

nItem - [in] This is the index of the item in the Parameter. This is a zero based number and thus must be in the range 0 to one less that the value returned by GetParameterLength.

pValue - [out] A pointer to a Variant object. The value returned here is the data stored at the given item number in the parameter. The language you are programming in will have functions to find out what exactly is stored (you could also use GetParameterType to figure out what to expect). There will also be conversion functions to convert this value to other data types as required.

Remarks

This function will report an error if the file has been opened in a mode less than 2 (See the Open function).

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nLength = itf.GetParameterLength(nIndex)
For I = 0 To nLength-1
      sValue = CStr(itf.GetParameterValue(nIndex, I))
Next I
   This set of calls will allow you to get the value of
   each item in the ANALOG:LABELS parameter as a string.
```

Set parameter name

This function sets the name of the specified parameter.

HRESULT SetParameterName (int nIndex, BSTR sName, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

sName - [in] This is the new name for the parameter. This name should not belong to any other parameter within the group.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2023 – The parameter name is empty.

2025 – The parameter name already exists within the group.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nSuccess = itf.SetParameterName(nIndex, “LABELS1”)
   This set of function calls will modify the name of
   the ANALOG:LABELS parameter to ANALOG:LABELS1.
```

Set parameter number

This sets the internal number used to refer to a parameter within the C3D parameter section.

HRESULT SetParameterNumber (int nIndex, int nNumber, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

nNumber - [in] This is the new number for the parameter.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2024 – The parameter number is not greater than zero.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

nSuccess = itf.SetParameterNumber(nIndex, 6)

   This set of function calls will modify the number of
   the ANALOG:LABELS parameter.  Note that when you do
   this the parameter will no longer belong to the
   ANALOG group but to whatever group is pointed to by
   the new number you assign.
```

Set parameter status

This sets the status (locked or unlocked) of the specified parameter.

HRESULT SetParameterLock (int nIndex, BYTE byLock, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

byLock - [in] This is the lock status of the parameter.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nSuccess = itf.SetParameterStatus(nIndex, 1)
   This set of function calls will lock the
   ANALOG:LABELS parameter.
```

Set parameter description

This function sets the description string for the specified parameter.

HRESULT SetParameterDescription (int nIndex, BSTR sDesc, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

sDesc - [in] This is the new description for the parameter.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

nSuccess = itf.SetParameterDescription(nIndex, “Labels
for analog channels”)

   This set of function calls will change the
   description of the ANALOG:LABELS parameter.
```

Set parameter type

This function sets the storage format (type) for the specified parameter.

HRESULT SetParameterType (int nIndex, int nType, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

nType - [in] This is the new data type in the parameter. This must be one of the valid data types.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

If you change the type of the parameter, the data contained in the parameter will be modified from the old type to the new type and you may not be able to get the old data back unless you store it elsewhere in your program.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2026 – Incorrect parameter type.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

nSuccess = itf.SetParameterType(nIndex, 2)

   This set of function calls will change the data type
   of this parameter to Integer.  Since the old data was
   of character type, there is no valid conversion and
   all of the data would be lost.
```

Set parameter value

This function sets the value of the specified parameter.

HRESULT SetParameterValue (int nIndex, int nItem, VARIANT vValue, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

nItem - [in] This is the zero based index of the item within the parameter that you want to change. This must be in the range 0 to one less that the value returned by GetParameterLength.

vValue - [in] This is the new value that you want to use. You must make sure that you supply the correct type of value based on the type of data stored in the parameter.

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

By default the C3Dserver will check that the first six characters of a new parameter name are unique within the group unless StrictParameterChecking has been disabled. Attempts to change the name of a parameter that contain the same six first six characters as an existing parameter within the same group will fail unless you first call the function StrictParameterChecking to disable this check.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

sValue = “LTIB”

nSuccess = itf.SetParameterValue(nIndex, 0, sValue)

   This will change the value of the first item of the
   ANALOG:LABELS parameter to LTIB.

nIndex = itf.GetParameterIndex(“ANALOG”, “USED”)

nChannels = 10

nSuccess = itf.SetParameterValue(nIndex, 0, nChannels)

   This will set the number of analog channels that have
   been used to 10.
```

Create parameter

This function creates a parameter with a specific value.

HRESULT AddParameter (BSTR sName, BSTR sDesc, BSTR sGroup, BYTE byLock, int nType, int nNumDim, VARIANT vDimList, VARIANT vData, int *pIndex)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

sName - [in] The name of the parameter you want to add. A parameter with a similar name should not exist in sGroup

sDesc - [in] The description for the parameter.

sGroup - [in] The Group to which the parameter should be added. This should be a valid group.

byLock - [in] The lock status of the parameter.

NType - [in] The type of data that is going to be stored

- -1 : Character
- 1 : Byte
- 2 : Integer
- 4 : Floating point

nNumDim - [in] The number of dimensions. For character data, you need to remember that the first dimension defines the length of each character string and so you need to have nNumDim = 2 to store at least one string.

vDimList - [in] An array containing the dimensions. This array must have as many elements as you have specified in nNumDim.

vData - [in] This is an array containing the actual data items. If this is empty, the parameter will be filled with default values.

pReturn - [out] –1, if the parameter was not added, else the index of the new parameter.

Remarks

By default the C3Dserver will check that the first six characters of the new parameter name are unique within the group unless StrictParameterChecking has been disabled. Attempts to create parameter names that contain the same six first six characters as an existing parameter within the same group will fail unless you first call the function StrictParameterChecking to disable this check.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2031 – The specified group was not found.

2023 – The parameter name cannot be empty.

2025 – The parameter name already exists in the group.

Examples

To create a parameter with character data

```
vDim(0) = 4 'set this to the length of each data item

vDim(1) = 10 'set this to number of data items you
expect

nReturn = ITF.AddParameter("UNITS", "Video Calibration
Units","POINT", 0, -1, 2, vDim, vData)
```

When you are creating a parameter to hold character data, you must have at least two dimensions. The first dimension would specify the length of each data item while the following dimensions would be used to figure out the number of items. In addition, you must be careful to set the nNumDim argument correctly to reflect the number of dimensions you want.

When you are using arrays with this type of server its dimensioning needs to be done in two steps, one to create it using DIM as a Variant and then REDIM it with correct dimensions and also with correct data type. This is because the server is looking for a specific data type and not just a Variant.

For example

```
Dim DimNum As Variant, data As Variant
```

Then for a 2 dimension parameter to hold character data:

```
ReDim DimNum(2) As Integer

ReDim data(50) As String

DimNum(0) = 25 'the length of each char

DimNum(1) = 50 ' the number of items you want to store

data(0) = “some string”

data(1) = “some other string“
```

Delete parameter

This function deletes the specified parameter.

HRESULT DeleteParameter (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

pReturn - [out] 1 if the change was made, 0 if it was not made.

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nSuccess = itf.DeleteParameter(nIndex)
   This will delete the ANALOG:LABELS parameter.
```

Copy parameter

This function copies a parameter to a temporary parameter structure that is internal to the C3Dserver. It allows the programmer to easily implement an "undo" function writing software that modifies a parameter in some way.

HRESULT CopyParameter (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This parameter is the index of the parameter that is to be copied. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

pReturn - [out] 1 if the parameter was copied, 0 if it was not.

Remarks

This functions copies the parameter at the given index to a temporary parameter structure. Only one parameter can be stored at a time in such a structure. You can set and retrieve the contents of this parameter by passing –1 to all the parameter functions.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nSuccess = itf.CopyParameter(nIndex)
```

This will copy the entire ANALOG:LABELS parameter to a temporary location that is internal to the server. To get any attributes of this parameter you need to call the Parameter functions like GetParameterDescription with an index of –1.

Retrieve parameter

This function copies the contents of a previously stored temporary parameter structure to the specified parameter. When used with the CopyParameter function it allows the programmer to easily implement an "undo" function writing software that modifies a parameter in some way.

HRESULT RetrieveParameter (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter to which the temporary parameter is to be copied. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

pReturn - [out] 1 if the parameter was copied, 0 if it was not.

Remarks

This function copies the content of the temporary parameter to the parameter at the given index.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

nSuccess = itf.CopyParameter(nIndex)

// do anything else that you might want

// if you want the old ANALOG:LABELS back use the next
// line

nSuccess = itf.RetrieveParameter(nIndex)
```

This will copy the entire ANALOG:LABELS parameter to a temporary location that is internal to the server. Later when you want to put it back to ANALOG:LABELS, you just have to use this function with the correct index.

Remove parameter data

This function removes one or more items from the parameter data structure depending on the parameter dimensions.

HRESULT RemoveParameterData (int nIndex, int nItem, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

nItem - [in] This is the item from the last dimension of the parameter that will be removed. See Remarks for explanation.

pReturn - [out] 1 if the parameter was copied, 0 if it was not.

Remarks

This function removes item(s) from the parameter data. The number of items that will be removed will depend on the dimensions of the parameter. This function has been written so that you can remove one item, say when you delete an analog channel and you want to remove one item from the ANALOG:LABELS parameter. Suppose you deleted channel 17, you would call RemoveParameterData(ParIndex, 16). What this would do is remove the 17th label from ANALOG:PARAMETERS.

Suppose you have two force platforms. The FORCE_PLATFORM:CORNERS would have dimensions 3,4,2. If you wanted to remove the first force plate information, you would call RemoveParameterData(ParIndex, 0). This would actually remove 12 items of data from the parameter, which corresponds to the corners of the first force plate.

You need to remember that nItem refers to a item in the last dimension only.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)

nSuccess = itf.RemoveParameterData(nIndex,16)
```

This will remove the label associated with the 17th analog channel. The length of the parameter will decrease by one item.

```
nIndex = itf.GetParameterIndex(“FORCE_PLATFORM”,
“CORNERS”)

nSuccess = itf.RemoveParameterData(nIndex,1)
```

You would use this command if you had two or more force plates and you wanted to remove the second force plate corners information.

Add parameter data

This function adds one or more items to the parameter data structure depending on the dimensions of the parameter.

HRESULT AddParameterData (int nIndex, int nItems, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the index of the parameter. This is a zero based number and thus must be in the range 0 to (GetNumberParameters-1).

nItems - [in] This is the number of elements that should be added to the list dimension of the parameter. The last dimension value will be increased by nItems. The total number of parameter data added will depend on the previous dimensions. This value has to be greater than zero. See Remarks

pReturn - [out] 1 if the parameter was copied, 0 if it was not.

Remarks

This function adds item(s) to the parameter data. The number of items that will be added will depend on the dimensions of the parameter. This function has been written so that you can add one item, say when you add an analog channel and you want to remove add one item to the ANALOG:LABELS parameter.

This function will work only if there number of dimensions is greater than zero.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – An invalid index used.

2030 – An invalid value to increase the dimensions.

Example

```
nIndex = itf.GetParameterIndex(“ANALOG”, “LABELS”)
nSuccess = itf.AddParameterData(nIndex,5)
```

This will cause five more items to be added to this parameter. You might want to do this if you are adding more analog channels.

Get 3D frame range

This function allows you to read either the first or last 3D frame numbers from an open C3D file and will return the values stored in the TRIAL parameters if they exist.

HRESULT GetVideoFrame (int *nStartEnd*, int * *nFrame*)

Return Value

A standard HRESULT value.

Parameters

nStartEnd - [in] This is an integer value which tells the function if you want to retrieve the start frame or the end frame. A value of 0 gives you the start frame while a value of 1 gives you the end frame.

nFrame - [out] A pointer to an integer. The value returned here is value of the start or end frame of point data that has been collected. This value is either from the header or from the TRIAL parameters.

Remarks

The value returned here in most cases will be from the header of the C3D file. However there are certain files in which the actual start and end frames are stored in the TRIAL:ACTUAL_START_FIELD and TRIAL:ACTUAL_END_FIELD parameters respectively. If the server finds these parameters, it will return the value from those parameters rather than from the header. These parameters allow you to overcome the 2-byte restriction on the start and end frame when they are stored in the header that otherwise limits the maximum frame number to 65535.

Example

```
nStartFrame = itf.GetVideoFrame(0)
nEndFrame = itf.GetVideoFrame(1)
```

```
This gives you the first frame of video data in the
nStartFrame variable and the last frame in the
nEndFrame variable.  This information is usually
taken from the header record of the file.  However,
if the TRIAL:ACTUAL_START_FIELD and
TRIAL:ACTUAL_END_FIELD parameters are present, the
values from those parameters are used.
```

3D and Analog Data functions

The 3D and analog data functions enable the user to read and write information from the 3D data block within the C3D file, as well as providing the ability to add and remove frames of data from the C3D file.

Read analog data sample

Reads, and optionally scales, an analog data sample from the open C3D file.

HRESULT GetAnalogData (int *nChannel*, int *nFrame*, int *nSubFrame*, BYTE *byScaled*, float *fOffset*, float *fScale*, BYTE *byUseScale*, VARIANT * *pData*)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetAnalogChannels-1).

nFrame - [in] This is the video frame number whose data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

nSubFrame - [in] This is the sub frame of the video frame number whose data you want to retrieve. This frame number should lie between 1 and GetAnalogVideoRatio (both inclusive)

byScaled - [in] Indicates whether you want scaled data or not. A value of 0 will give you un-scaled data while a value of 1 will give scaled data.

fOffset - [in] This is the offset value that should be used for scaling. If you want the server to find this on its own, set *byUseScale* to 0. If you want to provide your offset set *byUseScale* to 1. Either way if you want scaled data you have to set *byScaled* to 1.

fScale - [in] This is the scale value that should be used for scaling. If you want the server to find this on its own, set *byUseScale* to 0. If you want

to provide your own scale set *byUseScale* to 1. Either way if you want scaled data you have to set *byScaled* to 1.

pData - [out] A pointer to a Variant object. The value returned here is the data stored at the given frame and sub frame. The language you are programming in will have functions to find out what exactly is stored (you could also use GetParameterType to figure out what to expect). There will also be conversion functions to convert this value to other data types as required. The value stored here will be either an integer or a single precision floating-point number.

Remarks

Analog data in the C3D format is stored as integer values even if the actual file is a floating point (REAL) formatted file. The analog data format is fully described in the C3D User Guide.

The data is returned un-scaled if byScaled is set to 0. Under these circumstances, the value in the parameters, byUseScale, fScale and fOffset are not used.

The scaled data is returned if byScaled is set to 1. Under these circumstances, if byUseScale is 0, the application calculates it scale and offset values. If byUseScale is set to 1, the values in the fScale and fOffset parameters are used for scaling. The scaled value is calculated as:
*pData = (rawvalue – fOffset) * fScale*
where
fOffset – value in ANALOG:OFFSET parameter
fScale - value in ANALOG:SCALE parameter * value in ANALOG:GEN_SCALE parameter

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
fOffset = 0
fScale = 1
nSelChannel = 0
nFirst = itf.GetVideoFrame(0) ' FIrst Video Frame
nLast = itf.GetVideoFrame(1) ' Last Video Frame
nRatio = itf.GetAnalogVideoRatio ' Analog / Video Ratio
nFrames = (nLast - nFirst + 1) * nRatio ' Total frames
ReDim data(nFrames) As Variant
iCnt = 0
For i = nFirst To nLast ' Video Frames
 For j = 1 To nRatio ' Sub Frames
 data(iCnt) = itf.GetAnalogData(nSelChannel, i, j, 0,
fOffset, fScale, 0)
 iCnt = iCnt + 1
 Next j
Next i
```

The code fragment shown above will put all the data in channel 1 into the array data in unscaled form.

Read analog data range

Reads, and optionally scales, a range of analog data samples from the open C3D file.

HRESULT GetAnalogDataEx (int nChannel, int nStart, int nEnd, BYTE byScaled, float fOffset, float fScale, BYTE byUseScale, VARIANT * pData)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetAnalogChannels-1).

nStart - [in] This is the first frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

nEnd - [in] This is the last frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive). This should be greater than or equal to nStart.

byScaled - [in] Indicates whether you want scaled data or not. A value of 0 will give you un-scaled data while a value of 1 will give scaled data.

fOffset - [in] This is the offset value that should be used for scaling. If you want the server to find this on its own, set *byUseScale* to 0. If you want to provide your offset set *byUseScale* to 1. Either way if you want scaled data you have to set *byScaled* to 1.

fScale - [in] This is the scale value that should be used for scaling. If you want the server to find this on its own, set *byUseScale* to 0. If you want to provide your own scale set *byUseScale* to 1. Either way if you want scaled data you have to set *byScaled* to 1.

pData - [out] A pointer to an array of Variant objects. The value returned here is the data stored at the given frame and sub frame. The language you are programming in will have functions to find out what exactly is stored (you could also use GetParameterType to figure out what to expect). There will also be conversion functions to convert this value to other data types as required. The value stored here will be an array of integers or a single precision floating-point numbers. If you are asking for scaled values then the returned values will be floating point numbers. If you are asking for unscaled values then the returned values will be integer values if the file contains integer data or floating point values if the file contains floating point data.

Remarks

Analog data in the C3D format is stored as integer values even if the actual file is a floating point (REAL) formatted file. The analog data format is fully described in the C3D User Guide.

The data is returned un-scaled if byScaled is set to 0. Under these circumstances, the value in the parameters, byUseScale, fScale, and fOffset are not used.

The scaled data is returned if byScaled is set to 1. Under these circumstances, if byUseScale is 0, the application calculates its scale and offset values. If byUseScale is set to 1, the values in the fScale and fOffset parameters are used for scaling. The scaled value is calculated as:
*pData = (rawvalue – fOffset) * fScale*
where
fOffset – value in ANALOG:OFFSET parameter
fScale - value in ANALOG:SCALE parameter * value in ANALOG:GEN_SCALE parameter

The number of items returned is based on the nStart and nEnd values. If nStart and nEnd are equal, one video frame is returned. However, one video frame contains at Analog to Video ratio number of analog frames, which is the number of frames that will be returned. In general the number of frames will be:

*(nEnd – nStart + 1) * Analog to Video Ratio*

The array that is returned in a 0 based array. For a given video frame, all the analog frames are written and then the next video frame is written. For example if you want to get frame 1 and 2 from channel 4 in a file with AV Ratio as 3, you would use the commands,

```
Dim data As Variant
data = GetAnalogDataEx(3, 1, 2, 0,0,1,0)
```

data[0] would be frame 1 sub frame 1, then [1,2], [1,3], [2,1], [2,2], [2,3].

You can find out the size of the array data at runtime using the following command:

```
Size = UBound(data) – Lbound(data) + 1
```

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
fOffset = 0
fScale = 1
nSelChannel = 0
nFirst = itf.GetVideoFrame(0) ' FIrst Video Frame
nLast = itf.GetVideoFrame(1) ' Last Video Frame
nRatio = itf.GetAnalogVideoRatio ' Analog / Video Ratio
nFrames = (nLast - nFirst + 1) * nRatio ' Total frames
data = itf.GetAnalogDataEx(nChannel, nFirst, nLast, 0,
fOffset, fScale, 0)
```

The array data will now contain all of the data in the 1st channel.

Write analog data

Writes a new analog data sample value to the open C3D file.

HRESULT SetAnalogData (int nChannel, int nFrame, int nSubFrame, VARIANT vData, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetAnalogChannels-1).

nFrame - [in] This is the video frame number whose data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

nSubFrame - [in] This is the sub frame of the video frame number whose data you want to retrieve. This frame number should lie between 1 and GetAnalogVideoRatio (both inclusive)

vData - [in] This is the new value you want to use for the frame. The value you supply must be an un-scaled value. If the data file contains floating point data then this value can be a floating-point value

pReturn - [out] 1 if the data was written, 0 if it was not.

Remarks

Analog data in the C3D format is stored as integer values even if the actual file is a floating point (REAL) formatted file. The analog data format is fully described in the C3D User Guide.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2012 – Invalid frame number

2013 – Invalid channel number.

Example

```
Dim vData As Variant
vData = CInt(txtValue) ‘ get data from some text box
m_itf.SetAnalogData(1, 10, 0, vData)
```

This sets the data in the first sub frame of the tenth frame of the second channel in the file.

Write Analog Data Array

Write an array of analog data to a C3D file.

HRESULT SetAnalogDataEx (int nChannel, int nFrame, VARIANT vData, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetAnalogChannels-1).

nFrame - [in] This is the starting video frame number whose data you want to set. This frame number must lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive). The analog data will be written into the analog samples starting from this video frame untill all the values in the array vData have been exhausted.

vData - [in] This is the new values you want to use for the channel. The value you supply must be an un-scaled value. If the C3D data file contains floating point data then this value can be a floating-point value which will be converted and stored in the C3D file as an integer value. This parameter should contain an array of values with the number of values equal to the number of analog samples that you want to change.

pReturn - [out] 1 if the data was written, 0 if it was not.

Remarks

Analog data in the C3D format is stored as integer values even if the actual file is a floating point (REAL) formatted file. The analog data format is fully described in the C3D User Guide.

Added in Version 1.124, this function writes analog data into the C3D file to using the video frame as a reference. Analog data may contain multiply samples per video frame so writing analog data to a file with 10 analog samples per video frame will require an array ten times larger than the number of video frames to be written.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2012 – Invalid frame number

2013 – Invalid channel number.

2034 – Incorrect Data type

Example

The following code will write 100 analog values starting at video frame 20 in channel 0. If the analog sample rate is the same as the video frame rate then one sample will be written for each video frame – thus the file will need to be at least 120

frames long (20+100). If the analog sample rate is 10 samples per video frame then the file will need to be 30 frames long (20 +100/10).

```
Dim nTemp As Integer = 100
Dim vData(nTemp) As Integer
For i = 0 To nTemp – 1
      vData(i) = … ‘ set some value
Next I
nRet = SetAnalogDataEx(0,20,vData)
```

Change the Analog to Video Ratio

Allows you to set the analog to video ratio in C3D files that do not contain any analog channels.

HRESULT SetAVRatio (int nAVRatio, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

NAVRatio - [in] This is the value of the analog to video ratio that you want to set. This has to be greater than or equal to zero.

PReturn - [in] The return value is 1 if the operation was successful or 0 if there was some error..

Remarks

This function sets the value of the analog to video ratio in the header of the C3D file and must be used prior to creating analog channels in a C3D file. Note that you can not use this function to change the analog sample rate of a C3D file that contains analog data – it must be used before the analog channels are initially created. This function will also change the value of the ANALOG:RATE parameter to the correct value if this parameter already exists in the C3D file.

Possible Error Values

2057 – If there are existing analog channels this error is returned.

2058 – If you enter a value for the nAVRatio parameter that is lesss than zero

Example

```
nReturn = itf.SetAVRatio (10)
```

This call will set the analog to video ratio to 10 so that when analog channels are created they will have an analog sample rate of 10 analog values per analog channel for each video frame.

Create analog channel

Creates a new analog data channel.

HRESULT AddAnalogChannel (int *pChannelIndex)

Return Value

A standard HRESULT value.

Parameters

pChannelIndex - [out] The index of the analog channel added. –1, if the channel could not be added.

Remarks

The ANALOG:USED parameter is adjusted to reflect the correct number of analog channels. You need to make changes to any other parameters that might be affected.

The C3D format allows one sample rate for all analog channels. As a result added a new analog channel to a C3D file that already contains analog channels will cause the new channel to be created with the same sample rate as the existing analog channels. The analog sample rate (in terms of analog samples per video frame) can only be set (using the *SetAVRatio* function) before the first analog channel is added to a C3D file that contains only video channels.

The new channel will be populated with the value stored in the ANALOG:OFFSET parameter so that the initial values will be zero. As a result, it is recommended that all the required analog parameters such as OFFSET, SCALE etc are created prior to calling *AddAnalogChannel*.

Possible Error Values

2015 – If the data in the file was not read in the first place.

Example

```
nNewChannelIndex = m_itf.AddAnalogChannel
```

This creates a new analog channel and returns the index of that channel. Please note that you will need to make changes to all affected parameters.

Delete analog channel

Deletes an analog channel from the open C3D file.

HRESULT DeleteAnalogChannel (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the zero based index of the channel must be in the range 0 to (GetAnalogChannels-1).

pReturn - [in] 1 if the channel was deleted, 0 if it was not

Remarks

The ANALOG:USED parameter is adjusted to reflect the correct number of analog channels. You need to make changes to any other parameters that might be affected.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2013 – Invalid channel number.

Example

```
nSuccess = m_itf.DeleteAnalogChannel(0)
```

This deletes the first analog channel. You will need to make changes to all affected parameters.

Delete analog channel with parameters

Deletes an analog channel from the open C3D file. Also deletes data from the parameters related to the analog channel that was deleted. This function will not allow you to delete any of the Force Platform channels.

HRESULT DeleteAnalogChannelWithParameter (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the zero based index of the channel must be in the range 0 to (GetAnalogChannels-1).

pReturn - [out] 1 if the channel was deleted, 0 if it was not

Remarks

The ANALOG:USED parameter is adjusted to reflect the correct number of analog channels.

The ANALOG:LABELS, ANALOG:DESCRIPTIONS, ANALOG:SCALE, ANALOG:OFFSET, ANALOG:UNITS parameters are modified by deleting the entry for the channel that was deleted.

The channel numbers that are found in the FORCE_PLATFORM:CHANNELS parameter are also updated appropriately.

You need to make changes to any other parameters that might be affected.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2013 – Invalid channel number.

Example

```
nSuccess = m_itf.DeleteAnalogChannelWithParameter(0)
```

This deletes the first analog channel.

Get Force Data

Calculate the XYZ force data for a given force plate.

HRESULT GetForceData (int *nCord* , int *nFP* , int *nStart* , int *nEnd* , VARIANT * *pData*)

Return Value

A standard HRESULT value.

Parameters

nCord [in] This is the force data that you want to retrieve. The valid values are: 0 = FX, 1 = FY and 2= FZ

nFP [in] This is the zero based value of the force platform for which the force data needs to be calculated. This should be >=0 and <= (Number of FP – 1)

nStart [in] This is the first frame for which data needs to be calculated. This needs to be sent in terms of **video frames** and is a 1 based value.

nEnd [in] This is the last frame for which data needs to be calculated. This needs to be sent in terms of **video frames** and is a 1 based value. The value of nEnd cannot be less than the value of nStart. If both nStart and nEnd are –1, then all the frames of data are returned.

pData [out] A pointer to a Variant object. The value returned here is the force data for all the frames. The value stored here will be a single precision real number. The number of frames returned are calculated as: (nEnd – nStart + 1) * AnalogToVideoRatio

Remarks

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Possible Error Values

2015 – If the data in the file was not read in the first place.

2047 – The nFP value does not correspond to the number of force plates found.

2048 – Incorrect Force Component in nCord

2049 – Could not find the FORCE_PLATFORM:TYPE parameter in order to determine type of force platform.

2050 – Unknown force platform type. At this time force platforms have to be type 1, 2, 3 or 4.

2051 – Unsupported force platform. Type 1 Force Platforms are not supported.

2055 – Could not calculate force data. This could be because of memory allocation errors or could be because scaling for type 4 platforms could not be done correctly because of incomplete data.

Example

```
nFirst = itf.GetVideoFrame(0) ' First Video Frame
nLast = itf.GetVideoFrame(1) ' Last Video Frame
nRatio = itf.GetAnalogVideoRatio ' Analog / Video Ratio
nFrames = (nLast - nFirst + 1) * nRatio ' Total frames
data = itf.GetForceData(0, 0 , -1, -1)
```

The code fragment shown above will calculate FX values for all the frames in of the first force platform.

Get Moment Data

Get the moment data in the global coordinate system for a given force plate.

HRESULT GetMomentData (int *nCord* , int *nFP* , int *nStart* , int *nEnd* , VARIANT * *pData*)

Return Value

A standard HRESULT value.

Parameters

nCord [in] This is the moment data that you want to retrieve. The valid values are: 0 = DX, 1 = DY and 2 = TZ

nFP [in] This is the zero based value of the force platform for which the moment data needs to be calculated. This should be >= 0 and <= (Number of FP –1)

nStart [in] This is the first frame for which data needs to be calculated. This needs to be sent in terms of **video frames** and is a 1 based value.

nEnd [in] This is the last frame for which data needs to be calculated. This needs to be sent in terms of **video frames** and is a 1based value. The value of nEnd cannot be less than the value of nStart. If both nStart and nEnd are –1, then all the frames of data are returned.

pData [out] A pointer to a Variant object. The value returned here is the moment data for all the frames. The value stored here will be a single precision real number. The number of frames returned are calculated as: (nEnd – nStart + 1) * AnalogToVideoRatio

Remarks

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Possible Error Values

2015 – If the data in the file was not read in the first place.

2047 – The nFP value does not correspond to the number of force plates found.

2048 – Incorrect Force Component in nCord

2049 – Could not find the FORCE_PLATFORM:TYPE parameter in order to determine type of force platform.

2050 – Unknown force platform type. At this time force platforms have to be type 1, 2, 3 or 4.

2051 – Unsupported force platform. Type 1 Force Platforms are not supported.

2056 – Could not calculate moment data. This could be because of memory allocation errors or could be because scaling for type 4 platforms could not be done correctly because of incomplete data.

Example

```
nFirst = itf.GetVideoFrame(0) ' First Video Frame
nLast = itf.GetVideoFrame(1) ' Last Video Frame
nRatio = itf.GetAnalogVideoRatio ' Analog / Video Ratio
nFrames = (nLast - nFirst + 1) * nRatio ' Total frames
data = itf.GetMomentData(0, 0 , -1, -1)
```

The code fragment shown above will calculate DX values for all the frames in of the first force platform.

```
data = itf.GetMomentData(1, 0 , -1, -1)
```

The code fragment shown above will calculate DY values for all the frames in of the first force platform.

```
data = itf.GetMomentData(2, 0 , -1, -1)
```

The code fragment shown above will calculate TZ values for all the frames in of the first force platform.

Zero analog data

Perform baseline correction of analog data – several methods are supported, allowing baseline correction for any type of analog channel data.

HRESULT Zero Analog Channels (int *nChannel* , int *nMethod* , int *nFrames* , int * *pReturn*)

Return Value

A standard HRESULT value.

Parameters

nChannel [in] This is the zero based index of the channel must be in the range 0 to GetAnalogChannels -1.

nMethod [in] This is the method that must be used to zero the analog channel. The following are valid values:
1. Averages all the data in the channel
2. Uses the FORCE_PLATFORM:ZERO parameter to zero the data.
3. Uses nFrames frames from the start of the file.
4. Uses nFrames frames from the end of the file.

nFrames [in] This parameter is looked at only if nMethod is 3 or 4. It specifies the number of **video** frames that should be used for averaging.

pReturn [out] This is the return value and is 1 if the operation was performed correctly.

Remarks

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

An average value is calculated based on the method used. The difference of this value from the analog offset is found and that difference is then removed from all the data in the channel. Thus the actual data values are modified – this function does not change the ANALOG:OFFSET value.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2013 – Invalid Channel.

2052 – Incorrect method supplied to zero the analog channel.

2053 – Incorrect number of frames supplied to zero the analog channel.

Example

```
data = itf.ZeroAnalogChanenl(1, 3 , 25)
```

The code fragment shown above will adjust the first analog channel data based on the first 25 frames (video) of data in the channel.

Read 3D point

Reads a single 3D point value from the open C3D file.

HRESULT GetPointData (int *nChannel*, int *nCord,* int *nFrame*, BYTE *byScaled*, VARIANT * *pData*)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

nCord - [in] This is the coordinate for which you want the data. This can be 0, 1, or 2.

nFrame - [in] This is the video frame number whose data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

byScaled - [in] Indicates whether you want scaled data or not. A value of 0 will give you un-scaled data while a value of 1 will give scaled data. The value needed for scaling is taken from the header of the C3D file if the data is stored in Integer format. If the data is stored in Floating-point format, there is no need to scale the data.

pData - [out] A pointer to a Variant object. The value returned here is the data stored at the given. The language you are programming in will have functions to find out what exactly is stored (you could also use GetParameterType to figure out what to expect). There will also be conversion functions to convert this value to other data types as required. The value stored here will be either an integer or a single precision floating-point number.

Remarks

If the data is stored in the file is floating-point data, you will always get a scaled value from this function. However, if the data stored in integer, then you will get data based on the value of byScaled. This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
nFirst = itf.GetVideoFrame(0) ' Get First Frame
nLast = itf.GetVideoFrame(1) 'Get Last Frame
nFrames = (nLast - nFirst + 1)
For j = 0 To 2
iCnt = 0
ReDim data(nFrames) As Variant
For i = nFirst To nLast
```

```
 data(iCnt) = itf.GetPointData(0, j, i, 0)
 iCnt = iCnt + 1
 Next i
Next j
```

The code shown above will get all of the video data from the first marker into the array data. Note that only one coordinate of marker will be held in data at a given time.

Read 3D point range

Reads a range of 3D point values.

HRESULT GetPointDataEx (int *nChannel*, int *nCord,* int *nStart,* int *nEnd,* BYTE *byScaled,* VARIANT * *pData)*

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints -1).

nCord - [in] This is the coordinate for which you want the data. This can be 0, 1, or 2.

nStart - [in] This is the first frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

nEnd - [in] This is the last frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive). This should be greater than or equal to nStart.

byScaled - [in] Indicates whether you want scaled data or not. A value of 0 will give you un-scaled data while a value of 1 will give scaled data. For files in which data is stored as floating-point numbers, this value is not used. Data will always be scaled.

pData - [out] A pointer to an array of Variant objects. The value returned here is the data stored at the frames you have requested. The language you are programming in will have functions to find out what exactly is stored (you could also use GetParameterType to figure out what to expect). There will also be conversion functions to convert this value to other data types as required. The value stored here will be an array of integers or a single precision floating-point numbers.

Remarks

The number of items returned is based on the nStart and nEnd values. If nStart and nEnd are equal, one video frame is returned. In general the number of frames will be:

nEnd – nStart + 1

The array that is returned in a 0 based array. The data is stored one frame after another. For example if you want to get frame 10 to 20 from marker 4, Y coordinate in a file, you would use the commands,

```
Dim data As Variant
data = GetPointDataEx(3,1, 10, 20, 0)
```

data[0] would be frame 10, data[1] frame 11 and so on. You can find out the size of the array data at runtime using the following command:

```
Size = UBound(data) – Lbound(data) + 1
```

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
nFirst = itf.GetVideoFrame(0) ' Get First Frame
nLast = itf.GetVideoFrame(1) 'Get Last Frame
nFrames = (nLast - nFirst + 1)
dataX = itf.GetPointDataEx(0, 0, nFirst, nLast, 0)
dataY = itf.GetPointDataEx(0, 1, nFirst, nLast, 0)
dataZ = itf.GetPointDataEx(0, 2, nFirst, nLast, 0)
```

Here dataX, dataY, and dataZ will hold the data from the three coordinates of the first marker.

Read 3D residual range

Returns the 3D residual values for the specified range of data.

HRESULT GetPointResidualEx (int *nChannel*, int *nStart,* int *nEnd,*, VARIANT * *pData)*

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

nStart - [in] This is the first frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

nEnd - [in] This is the last frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive). This should be greater than or equal to nStart.

pData - [out] A pointer to an array of Variant objects. The value returned here is the residual data stored at the frames you have requested. The language you are programming in will have functions to find out what exactly is stored (you could also use GetParameterType to figure out what to expect). There will also be conversion functions to convert this value to other data types as required. The value stored here will be an array of single precision floating-point numbers.

Remarks

The number of items returned is based on the nStart and nEnd values. If nStart and nEnd are equal, one video frame is returned. In general the number of frames whose residual is returned will be:

nEnd – nStart + 1

The array that is returned in a 0 based array. The data is stored one frame after another. For example if you want to get residual for frames 10 to 20 from marker 4 in a file, you would use the commands,

```
Dim data As Variant
data = GetPointResidualEx(3, 10, 20)
```

data[0] would be frame 10, data[1] frame 11 and so on. You can find out the size of the array data at runtime using the following command:

```
Size = UBound(data) – Lbound(data) + 1
```

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
nFirst = itf.GetVideoFrame(0) ' Get First Frame
```

```
nLast = itf.GetVideoFrame(1) 'Get Last Frame
nFrames = (nLast - nFirst + 1)
Residual = itf.GetPointResidualEx(0, nFirst, nLast)
```

Here Residual will hold the all the residuals of the first marker.

Read 3D residual

Returns the 3D residual value of a specific point.

HRESULT GetPointResidual (int *nChannel*, int *nFrame*, float* *fValue)*

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

nFrame - [in] This is the video frame number whose data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

fValue - [out] A pointer to a single precision floating-point object. The value returned here is the residual value at the given frame.

Remarks

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
nFirst = itf.GetVideoFrame(0) ' Get First Frame
nLast = itf.GetVideoFrame(1) 'Get Last Frame
nFrames = (nLast - nFirst + 1)
iCnt = 0
ReDim data(nFrames) As Variant
For i = nFirst To nLast
data(iCnt) = itf.GetPointResidual(0, i)
iCnt = iCnt + 1
Next i
```

The code shown above will get all of the residuals from the first marker into the array data.

Read 3D mask

Returns the camera mask associated with a single 3D data point.

HRESULT GetPointMask (int *nChannel*, int *nFrame,* BSTR* *pMask)*

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

nFrame - [in] This is the video frame number whose data you want to retrieve. This frame number should lie between GetVideoFrame (0) and GetVideoFrame(1) (both inclusive).

pMask - [out] A pointer to a string. The value returned here is a string, which shows you the camera mask for the seven cameras. The left most character is the first camera. A value of '0' at a position indicates that the camera did not see the marker while a value of '1' indicates that the marker was seen.

Remarks

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

```
nFirst = itf.GetVideoFrame(0) ' Get First Frame
 nLast = itf.GetVideoFrame(1) 'Get Last Frame
 nFrames = (nLast - nFirst + 1)
 iCnt = 0
 ReDim data(nFrames) As Variant
 For i = nFirst To nLast
 data(iCnt) = itf.GetPointMask(0, i)
 iCnt = iCnt + 1
 Next i
```

The code shown above will get all of the masks from the first marker into the array data.

Read 3D mask range

Returns the camera mask associated with a range of 3D data points.

HRESULT GetPointMaskEx (int *nChannel*, int *nStart,* int *nEnd,*, VARIANT * *pData)*

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints -1).

nStart - [in] This is the first frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

nEnd - [in] This is the last frame of data you want to retrieve. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive). This should be greater than or equal to nStart.

pData - [out] A pointer to an array of Variant objects. The value returned here is the camera mask at the frames you have requested. The value stored here will be an array of strings. Each string shows you the camera mask for the seven cameras. The left most character is the first camera. A value of '0' at a position indicates that the camera did not see the marker while a value of '1' indicates that the marker was seen

Remarks

The number of items returned is based on the nStart and nEnd values. If nStart and nEnd are equal, one video frame is returned. In general the number of frames whose residual is returned will be:

nEnd – nStart + 1

The array that is returned in a 0 based array. The data is stored one frame after another. For example if you want to get the mask for frames 10 to 20 from marker 4 in a file, you would use the commands,

```
Dim data As Variant
data = GetPointMaskEx(3, 10, 20)
```

data[0] would be the mask at frame 10, data[1] at frame 11 and so on. You can find out the size of the array data at runtime using the following command:

```
Size = UBound(data) – Lbound(data) + 1
```

This function will report an error if the file has been opened in a mode less than 3 (See the Open function).

Example

```
nFirst = itf.GetVideoFrame(0) ' Get First Frame
```

```
nLast = itf.GetVideoFrame(1) 'Get Last Frame
nFrames = (nLast - nFirst + 1)
Mask = itf.GetPointMaskEx(0, nFirst, nLast)
```

Here Mask will hold the all the camera masks of the first marker.

Write 3D point

Write a single 3D data point value, residual or camera mask.

HRESULT SetPointData (int nChannel, int nCord, int nFrame, VARIANT vData, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

nCord - [in] This is the coordinate for the data that you wish to write. This can be 0, 1, or 2. If you want to set the residual you have to enter a value of 3 and if you want to set the camera mask, you have to set a value of 4.

nFrame - [in] This is the video frame number whose data you want to write. This frame number should lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive).

vData - [in] This is a Variant object. This should contain the value you want to store in the marker. The coordinate data should be unscaled, while the residual data should always be scaled. The camera mask should be a string with a maximum of seven characters, with each character being 0 or 1.

pReturn - [in] 1 if the data was written, 0 if it was not

Remarks

Possible Error Values

2015 – If the data in the file was not read in the first place.

2012 – Invalid frame number

2013 – Invalid channel number.

Example

```
Dim vData As Variant
Dim sTemp As String
sTemp = txtValue ‘ value from some text box
nChannel = 0
nFrame = 50
if this is coordinate information
vData = CInt(sTemp)
m_itf.SetPointData(nChannel, 0, nFrame, vData)
```

```
if this is residual information
vData = CSng(sTemp)
m_itf.SetPointData(nChannel, 3, nFrame, vData)
if this is a camera mask
sTemp = Trim(sTemp)
If Len(sTemp) <= 7
vData = sTemp
m_itf.SetPointData(nChannel, 4, nFrame, vData)
EndIf
```

Write 3D point Array

Write an array of a single co-ordinate value to a C3D file.

HRESULT SetPointDataEx (int nChannel, int nCord, int nFrame, VARIANT vData, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nChannel - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

nCord - [in] This is the co-ordinate that has to be changed. The three co-ordinates are 0, 1 and 2 (X, Y, Z). Use 3 for the residual and 4 for the camera mask.

nFrame - [in] This is the starting video frame number whose data you want to set. This frame number must lie between GetVideoFrame(0) and GetVideoFrame(1) (both inclusive). The data will be written starting from this frame until all the values in the array vData have been exhausted.

vData - [in] This is the new values you want to use for the channel. The value you supply must be an un-scaled value. If the data file contains floating point data then this value can be a floating-point value. This parameter should contain an array of values with the number of values equal to the number of frames you want to change. If you are setting the residual then the values have to be floating point values while if you are setting the camera mask the values have to be string values.

pReturn - [out] 1 if the data was written, 0 if it was not.

Remarks

Added in Version 1.124 to improve performance.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2011 – Invalid co-ordinate number

2012 – Invalid frame number

2013 – Invalid channel number.

2034 – Incorrect Data type

Example

The following code will write integer data into 100 video frames starting at frame 20 for the Y co-ordinate of the marker 0 point data channel. Note that the data channel must exist before data can be written to it (Goodes Law).

This function writes a single co-ordinate at a time and must be called five times to write complete *X, Y, Z, mask, residual* values, once for each co-ordinate, or associated camera mask and residual value.

```
Dim nTemp As Integer = 100
Dim vData(nTemp) As Integer
For i = 0 To nTemp - 1
      vData(i) = … ‘ set some value
Next i
nRet = SetPointDataEx(0,1,20,vData)
```

Create 3D point

Creates a new 3D data point.

HRESULT AddMarker (int *pMarkerIndex)

Return Value

A standard HRESULT value.

Parameters

pMarkerlIndex - [out] The index of the marker added. –1, if the marker could not be added.

Remarks

The POINT:USED parameter is adjusted to reflect the correct number of markers. You need to make changes to any other parameters that might be affected.

Possible Error Values

2015 – If the data in the file was not read in the first place.

Example

```
nIndex = itf.AddMarker
```

This creates a new marker. nIndex will hold the index of the new marker if it was created or –1 if it could not be created.

Delete 3D point

Deletes the specified 3D data point from the open C3D file.

HRESULT DeleteMarker (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

pReturn - [in] 1 if the channel was deleted, 0 if it was not

Remarks

The POINT:USED parameter is adjusted to reflect the correct number of markers. You need to make changes to any other parameters that might be affected.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2013 – Invalid channel number.

Example

```
nSuccess = itf.DeleteMarker(5)
   This call will delete the sixth marker in the file.
```

Delete 3D point with parameters

Deletes the specified 3D data point from the open C3D file. Also deletes data from the parameters related to the data point that was deleted.

HRESULT DeleteMarkerWithParameter (int nIndex, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nIndex - [in] This is the zero based index of the channel must be in the range 0 to (GetNumber3DPoints-1).

pReturn - [out] 1 if the channel was deleted, 0 if it was not

Remarks

The POINT:USED parameter is adjusted to reflect the correct number of markers.

The POINT:LABELS and POINT:DESCRIPTIONS parameters are modified by deleting the entry for the marker that was deleted.

You need to make changes to any other parameters that might be affected that are not mentioned above.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2013 – Invalid channel number.

Example

```
nSuccess = itf.DeleteMarkerWithParameter(5)
```

This call will delete the sixth marker in the file.

Add frames

Adds the specified number of frames to the open C3D file at the specified frame number.

HRESULT AddFrames (int nFrames, int nInsertAt, int *nReturn)

Return Value

A standard HRESULT value.

Parameters

nFrames - [in] This is the number of frames that is to be added. The total number of frames after addition of these frames should not exceed 65535.

nInsertAt - [in] This parameter controls the location where the frames are going to be inserted. The possible values are:
-1 : frames are inserted before the first frame
-2 : frames are inserted after the last frame
If any other positive value is passed, the frames are inserted at that frame. The value should be a valid frame number. This is a one-based number.

pReturn - [in] The number of frames present if successful, -1 if the operation was not performed.

Remarks

This function adds frames of data at a location determined by the nInsertAt parameter. The POINT:FRAMES parameter is adjusted to reflect the correct number of frames after this operation.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2036 – The frame limit was exceeded

2037 – The nInsertAt parameter was incorrect

Example

```
nTotalFrames = itf.AddFrames(121,-1)

   This call will add 121 frames at the start of the
   data in the file.

nTotalFrames = itf.AddFrames(121,-2)

   This call will add 121 frames after all the existing
   frames of data in the file.

nTotalFrames = itf.AddFrames(45,10)

   This call will insert 45 frames at the 10th frame of
   data in the file.
```

Delete frames

Deletes the specified frame range from the open C3D file.

HRESULT DeleteFrames (int nStartAt, int nNumFrames, int *pReturn)

Return Value

A standard HRESULT value.

Parameters

nStartAt - [in] This is the frame number at which the module should start deleting the frames. This is a 1-based number that should be within the valid frame values.

nNumFrames - [in] This is the number of frames to be deleted. This should not exceed the number of frames between nStartAt and the last frame.

pReturn - [in] The number of frames present if successful, -1 if the operation was not performed.

Remarks

This function adds frames of data at a location determined by the nInsertAt parameter. The POINT:FRAMES parameter is adjusted to reflect the correct number of frames after this operation.

Possible Error Values

2015 – If the data in the file was not read in the first place.

2037 – The nStartAt parameter was incorrect

2038 – Incorrect number of frames was specified.

Example

```
nTotalFrames = itf.DeleteFrames(10, 25)
```

This call will remove 25 frames starting from the 10th frame of data in the file. *nTotalFrames* will contain the number of frames that are left after this operation is complete. It will contain –1, if the operation could not be executed.

The way to safely delete all frames is:

```
FirstFrame = c3d.GetVideoFrame(0)
NumFrames = c3d.GetVideoFrames(1) - FirstFrame + 1
c3d.DeleteFrames(FirstFrame,NumFrames
```

Error Reports

The following error numbers may be returned by the C3Dserver to indicate a problem with the requested function.

Error	Description
2000	Could not open the selected file – e.g., the file could be locked by another user, it might not exist or not use the C3D format.
2001	Could not read the header information – e.g., the header may be corrupt or this is not a C3D file.
2002	Selected file is not a C3D file.
2003	Could not find the file format of the selected file – e.g., the file is probably corrupt?
2004	Could not find the location of first parameter record – e.g., possibly a corrupt C3D header record or incorrectly written file.
2005	No memory to create groups
2006	No memory to create parameters
2007	No memory to create data
2008	File not found in specified location
2009	File does not have Read permission – e.g., you do not have permission to read the file.
2010	File is not open.
2011	Invalid Index has been used.
2012	Invalid Frame Number.
2013	Invalid Channel Number.
2014	The C3Dserver has not been installed using the Motion Lab Systems setup program. Reinstall the C3Dserver using the Motion Lab Systems installation program – select the "compact" installation if you do not require the sample data files or documentation on the target system.
2015	This error is reported if you try to access data that has not been read. The data that is read is controlled by the nMode variable in the Open function.
2016	No more events can be added – e.g. the C3D file header event storage allocation is full.
2017	The event label cannot be empty.
2018	Invalid Event Time – you probably specified a time that is outside the 3D frame range stored in the C3D file.
2019	Incorrect value for group number.

2020	Invalid group name.
2021	Group number already exists.
2022	Group name already exists.
2023	Invalid parameter name.
2024	Parameter number has to be greater than zero.
2025	This parameter already exists within the group.
2026	Incorrect data type
2027	Conversion not possible to this data type – e.g., you cannot change a string to a numeric data type or vice versa.
2028	Incorrect number of dimensions specified
2029	Invalid index into the dimension list
2030	Invalid value for the dimension
2031	Could not find the selected group
2032	The DimList parameter needs to be an array
2033	Incorrect number of elements in DimList
2034	Incorrect data type passed to the function
2035	Could not delete analog channel – e.g., you are probably trying to delete a channel that does not exist.
2036	The frame limit of 65,535 frames has been exceeded by this request.
2037	Invalid start frame number.
2038	Incorrect number of frames specified.
2039	Could not delete marker – e.g., you have probably trying to delete a 3D marker/point that does not exist.
2041	The file name is empty. You will get this error if you try to save a New file without specifying a file name.
2042	The value of interpolation gap needs to be greater than or equal to zero.
2046	Incorrect sampling rate

Revision History

The policy of Motion Lab Systems is to release new versions of the C3Dserver to correct programming errors, incorporate new functions, and provide additional documentation.

Any application can perform a function call to determine the C3Dserver version number – it is strongly recommended that all applications written with the C3Dserver retrieve and display the current C3Dserver version number in the applications "about" dialog so that the user can determine which version of the C3Dserver is running. The following versions of the C3Dserver have been released:

Working Version

This working version released on 19 July 2010 and contains no changed to the C3Dserver DLL. Documentation revision and updated licensing conditions.

Version 1.144.0

Released on July 10, 2010, this version adds three new functions to aid with C3D file housekeeping; these are:

DeleteMarkerWithParameter – this deletes a specified 3D point and the associated POINT:LABEL and POINT:DESCRIPTION parameters values

DeleteAnalogChannelWithParameter – this deletes a specified analog channel and the associated ANALOG:SCALE parameter values. This function will not allow you to delete any of the Force Platform channels.

CompressParameterBlocks - this function ensures that any empty parameter blocks are removed when the C3D file is saved. This is useful if you have deleted a large number of parameters from the file.

Version 1.143.0

Released on April 6, 2010 this version of the C3Dserver will now check that the FORCE_PLATFORM parameters exist when a file is opened before trying to reference them internally. Previous versions of the C3Dserver will crash if they try to open a C3D file that does not contain a FORCE_PLATFORM group or a FORCE_PLATFORM:USED parameter.

Version 1.142.1

A documentation change in the manual to correct an error in the description of the **GetNumberParamters** function. This deletes incorrect references to the **GetParameterCount** function. The C3Dserver DLL is unchanged.

Version 1.142.0

Released on April 5, 2009 this version fixed a potential data overflow problem in the Fx force plate calculations.

Version 1.141.2

Cleanup and documentation changes in the manual and sample applications supplied with the C3Dserver to clearly explain the registered vs. unregistered status of the C3Dserver and document the various licensing modes. The sample code for the GetRegistrationMode function has changed, but the function itself and the C3Dserver DLL have not changed.

Version 1.141.1

Documentation change in the manual only. Corrected the example code in the GetNumberParameters function from *nParams = itf.GetParameterCount* to **nParams = itf.GetNumberParameters** which is correct. There is no GetParameterCount function and the C3Dserver DLL is unchanged.

Version 1.141

Released on February 22, 2008, this version corrects a problem with files that use the TRIAL:ACTUAL_START_FIELD and TRIAL:ACTUAL_END_FIELD parameters that results in an *Error 2007 No memory to create data* report.

Version 1.134

The GetAnalogData functions have been corrected to interpret the parameter FORCE_PLATFORM:ZERO in terms of 3D frames - previous versions treat this parameter incorrectly as analog samples. Applications that referenced this parameter may need to be checked.

Version 1.132.1

This release contains documentation revisions only. The C3Dserver DLL has not changed – this is a documentation change only and does not affect any existing code.

The Get Force Data function was incorrectly described in the manual:

The correct description is:

nCord [in] This is the force data that you want to retrieve. The valid values are: **0 = FX, 1 = FY and 2= FZ**

The Get Moment Data function was incorrectly described in the manual:

The correct description is:

nCord [in] This is the moment data that you want to retrieve. The valid values are: **0 = DX, 1 = DY and 2 = TZ**

Additional examples have been provided for the Get Moment Data function.

Version 1.132

Corrects an error the occurred when deleting all frames from a C3D file and prevented the last frame from being removed.

Version 1.131

The C3Dserver no longer requires a registration key before it will run in the "evaluation" mode. This means that the C3Dserver DLL can now be easily distributed with other applications in the "evaluation" mode simply by copying the DLL to the target system and installing it via regsvr32.

Added information to the manual about the functions that calculate forces and moments, as well as the ZeroAnalogData. Updated the documentation for the AddAnalogData function and fixed typo in the VBservertest application.

Added a .NET example project to the standard installation. The sample shows the registration name of the user when the application starts up. This is to show that the C3Dserver can be called up from within .net 2003 (i.e. Net Framework 1.1). If you are creating a new project then do the following from with the Visual Studio application:

1. Use the Project: Add Reference command and go to the COM tab in the resulting dialog.
2. Double click on the C3D Server entry. This will add a reference to the server library.
3. Then from within the code you use the library thus "Dim server As New C3DSERVERLib.C3D"

Version 1.129

Released on March 8, 2006 to modify the AddAnalogChannel function. When it creates a new analog channel it will now look for the channel OFFSET value and use this as the default data value. If the OFFSET parameter does not exist for the new channel then the function will use the data value from a lower numbered channel. If the OFFSET parameters can not be found then the channel is populated with 2048.

Added documentation to the manual for the *GetForceData*, *GetMomentData* and *ZeroAnalogData* functions that were added in version 1.100 but inexplicably, not documented in the manual.

Version 1.128

Released on October 10, 2005, this version adds a new function *SetAVRatio* that can be called to set the analog sample rate prior to the creation of analog data channels. Note that the analog sample rate must be set before analog channels are added to the C3D file – this function does not permit the user to change the analog sample rate of existing analog channels.

Version 1.125

Released on January 31, 2005, this version allows the user to control the C3D specification checking that is performed by the C3Dserver DLL with a new function called *SetStrictParameterChecking*. The C3D format specification requires that the first six characters of parameter and group names are unique - this is enforced by default in the C3Dserver DLL. If you need to use, or access, parameters that cannot pass this test, then you can use this function to disable strict checking.

Version 1.124

Released on January 24, 2005, this version adds two new functions that allow blocks of analog or point data to be written to the C3D file in a single operation. These functions are *SetAnalogDataEx* and *SetPointDataEx* – their use should considerably

speed up all C3D write operations, especially for anyone writing code with the evaluation version of the C3Dserver. This is the first version of the installation program that supports automatic updates from our FTP site.

Version 1.123

Released on October 18, 2004, this version allows C3D files to be opened when the file or media is read-only. Other changes have been made to the support files supplied with the C3Dserver to fix minor bugs – these changes were released with updated versions of 1.121 and 1.122 but did not change the C3Dserver DLL code.

Version 1.120

Released on August 1, 2003, this version adds support for the Vicon Motion Systems 16-bit ADC file data format when the parameter ANALOG:FORMAT is set to indicate the presence of 16-bit data.

Version 1.110

Released on September 6, 2002, this version corrects a bug in the way that the C3Dserver interprets requests to write analog data.

The *SetAnalogData* function has been changed to accept floating-point values in the *vData* variable. These values will be applied as floating point numbers if the storage format of the file is floating point, otherwise it will be converted to integer. Previous versions of the C3Dserver always converted the analog data to integer even if the file format was floating point.

Analog data returned by the *GetAnalogDataEx* function, when un-scaled values are requested, now depends on the data format of the file. If data is stored in floating point format then floating point values are returned, while if data is stored in integer format then integer data is returned.

The documentation supplied with the C3Dserver has been revised to match the same terms and C3D descriptions used in the companion C3D Format manual. Thus, terms such as “REAL” have been replaced with the more common term “floating-point” etc.

Version 1.100

Released on June17, 2002 as a result of suggestions made during the C3D User Group meeting earlier in the year. This version adds two new functions that produce force and moment data directly from the force plate using the C3D file parameters to determine the force pate TYPE etc. *GetForceData* generates Fx, Fy, and Fz while *GetMomentData* generates Mx, My, and Mz directly. In addition, the new *ZeroAnalogData* function allows the user to easily zero any type of analog data and includes several flexible options to accommodate different types of raw data. The distribution includes updates to the Visual Basic sample application to demonstrate the new functions.

This release of the C3Dserver is the first to offer comprehensive support for analog data from TYPE-4 force plates that support a cross-talk correction matrix.

This release was updated on August 20, 2002 to include additional documentation to demonstrate the use of the C3Dserver with Microsoft C++ Foundation Classes. This update adds C++ sample code to the installation for a small application that creates various example C3D files.

Version 1.015

Released on November 2nd, 2001 – this version adds functions to set interpolation length in the header and includes new functions to automate the creation new C3D files via a single call.

This release was updated on March 29, 2002 to include sample Visual Basic code to demonstrate each function. The C3Dserver DLL is unchanged.

Version 1.013

Released on October 17, 2001 – this is the first public first public release of the C3Dserver for evaluation via the C3D-L mail list. It includes full documentation and a source code for a simple C3D file editor written in Visual Basic.

Index